Keeping Fairness Alive

Torabi Dashti, M.

2008

Keeping Fairness Alive

Design and formal verification of optimistic fair exchange protocols

Mohammad Torabi Dashti

Printed by Ponsen & Looijen B.V.
ISBN 978-90-6464-220-3
IPA dissertation series 2008-07

Nederlandse Organisatie voor Wetenschappelijk Onderzoek

The work in this thesis has been carried out at the centre for mathematics and computer science (CWI), under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The research has been funded by the Dutch organisation for scientific research (NWO) in the context of the ACCOUNT project on accountability in electronic commerce protocols.

VRIJE UNIVERSITEIT

Keeping Fairness Alive

Design and formal verification of optimistic fair exchange protocols

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op woensdag 27 februari 2008 om 10.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Mohammad Torabi Dashti

geboren te Kordkoy, Iran

promotoren: prof.dr. W.J. Fokkink
prof.dr. J.C. van de Pol

Acknowledgements

This thesis is the result of my research in the last four years at CWI. In the following, I mention some of those who helped me in doing this research.

First comes Wan Fokkink, my supervisor and promotor. The liberty Wan gave me to follow my research interests and his constant encouragement were of great importance to me, both personally and professionally.

A large part of the results that I report in the thesis has been produced jointly with Jan Cederquist. Jan has been the first critic of my work, besides being a very supportive friend.

I would like to thank the members of the reading committee of my thesis, Bruno Crispo, Sandro Etalle, Joshua Guttman and Sjouke Mauw. Their feedback in many ways improved the quality of the thesis. I am also grateful to Jan Cederquist and Felix Freiling for participating in my promotion committee as opponent members.

Among other colleagues, my promotor Jaco van de Pol and Michael Weber answered a lot of my questions and commented on my raw ideas. I very much appreciate their enthusiasm. I attended several herfstdagen and lentedagen schools organised by IPA, in which I was a member of the PhD council for the last three years. I also participated in two courses at the Vrije Universiteit, one on type theory taught by Freek Wiedijk and one on distributed algorithms taught by Wan Fokkink. It was very kind of them to let me participate in these courses. I sincerely thank Paul Klint, the current head of our team, for his support in facilitating my stay at CWI in the last few months.

In collaboration with Anton, Bert, Hugo, Jaco, Jan, Jens, Jun, Ricardo, Simona, Sjouke, Srijith, Tom, Wan and Yanjing, I co-wrote a few papers, some of which presented in this thesis. I enjoyed the time we spent together discussing "work" at CWI and elsewhere.

A special gratitude goes to my parents, my brother Hesam and my sister Sanaz, with whom I shared my up and down times. I could not have led any kind of productive life in these four years without the huge amount of support that I received from my friends, particularly those residing in Amsterdam. I was lucky to have been surrounded by them.

Contents

Chapter 1

Introduction

> I don't read, but, I'll tell you one thing for sure: I wouldn't trust no words written down on no piece of paper, especially from no "Dickinson" out in the town of Machine.
>
> Jim Jarmusch's *Dead man* (1995)

You have probably seen products bearing *fair trade* labels in supermarkets. Fairness, there, refers to the ethical basis of trading those goods. In this thesis, however, by fairness we refer to the basic property of *atomicity*, i.e. either the seller receives the money and the buyer receives the goods, or none of them does so. This is thus a property of the means of the trade, rather than the trade itself. Our focus is on security protocols which facilitate (fair) exchange in electronic commerce.

In conventional commerce, deals are usually administered by law. If a customer pays for a product to a vendor and the vendor fails to deliver the product (as stated in their contract), then the customer can resort to litigation, which is enforceable by law. In electronic commerce, however, litigation is often not viable. This is because adequate laws to evaluate and judge based on electronic documents may be lacking, the exchange partners may not be governed by the same law (e.g. they may live in different countries) and, more importantly, the accountable real world party behind an electronic agent may not be traceable, cf. [San97].

The current practice of electronic commerce, therefore, heavily relies on trusted third parties. Most electronic commerce sites, for instance, offer little beyond browsing their catalogues, while contract signing and payment often consist of entering a credit card number. The trust in these sites is largely built upon the trust users have in the credit card companies, which keep records and in case of fraud, provide compensation. Achieving fairness in electronic commerce in fact turns out to be impossible if there is no presumed trust among the involved parties [EY80].

When there is a mediator who is trusted by all the exchange partners, conceptually, the items subject to exchange can be sent to the trusted entity and then he would distribute them if all the items arrive in time. Otherwise, he would simply abandon the exchange. This mechanism is inefficient and can be improved by reducing the involvement of the trusted party, such that he would need to play a role only when something goes amiss in the exchange. Such protocols are preferable when assuming that most exchange partners are honest and, thus, failed exchanges are infrequent (hence being *optimistic*). Design and formal analysis of optimistic fair exchange protocols constitute the content of this thesis.

Let us assume that Alice and Bob wish to exchange some data electronically. We note that in general one of the exchange partners, say Alice, receives what she wishes after Bob has received his desired item. Fairness thus partly refers to an event in the future: When Bob receives the money, Alice either has received the goods or will (inevitably) receive them in the future. Properties which stipulate the inevitability of an event are usually referred to as *liveness* properties, hence comes *keeping fairness alive*.

Overview

This thesis is the result of research performed in the context of the ACCOUNT project, which was funded by the Dutch organisation for scientific research (NWO). The ACCOUNT project addresses *accountability* in electronic commerce. According to Bella and Paulson, accountability "reduces the need for trust" [BP06]. They argue that "[classical] security protocols establish secure communications over insecure networks. Typically they ensure that no attacker can obtain sensitive information or impersonate another person. The protocol protects Alice and Bob, who trust one another, from hostile parties. This scenario is inappropriate when Alice does not even know Bob, let alone trust him. Purchasing goods over the Internet requires trusting the merchant with your credit card details, even if a protocol such as SSL protects against outsiders". Developing tools "to analyse accountability in existing e-commerce protocols" and designing "new protocols for electronic negotiation and payment" are the primary goals of the ACCOUNT project [CEF03]. This thesis can correspondingly be divided into two parts.

Part I: Designing fair exchange protocols

After introducing security protocols, in general, and fair exchange protocols, in particular in § 2, we present a fair *certified email* protocol, which is an instance of fair exchange, in § 3. A certified email protocol enables Alice to send an email to Bob in exchange for a receipt. The receipt is a proof that shows Bob has received the email. A fair certified email protocol guarantees fairness in this exchange: Bob receives the email iff Alice receives the receipt.

A unique feature of the proposed certified email protocol is that it relies on *key chains* to reduce the amount of storage that the trusted entity requires (recall that trusted entities unavoidably have a role in any optimistic fair exchange protocol). A key chain is a sequence of keys such that each key in the chain is derived by applying a certain function to the previous element. Using key chains in security protocols can perhaps be traced back to Lamport [Lam81].

This certified email protocol gives the reader a taste of the sort of the protocols that are to be analysed later. As a matter of fact, this protocol has been designed at a very late stage of our research, and has not yet been formally analysed. However, if the reader finds it difficult to believe that this protocol is secure, despite the informal justification that is provided, then he

or she is probably very well motivated to look into the formal verification of similar protocols, which is the topic of the second part of the thesis.

Another fair exchange protocol that we designed, and did formally verify, within the ACCOUNT project, is a fair *non-repudiation* protocol. Non-repudiation guarantees that an agent cannot deny having sent or received a message, if it has actually done so in the course of the protocol. To achieve this, protocol participants usually collect evidences, evidence of origin and evidence of receipt, which can later be presented to a judge. Such a protocol is fair iff these evidences are exchanged in a fair manner. See § 6.2.2 and also [CCT05].

Part II: Formal analysis of fair exchange protocols

Security protocols are notoriously difficult to design and fully understand. Formal verification can shed some light on how a security protocol behaves in the presence of a particular *intruder* model. The intruder model has to reflect the hostile environment in which the protocol is going to be deployed. If the intruder model is weaker than the real world intruder, the formal analysis is *unsound*. This means that provably secure protocols may easily be subverted in practice, as the model has underestimated the intruder's power. Conversely, an intruder model which is stronger than the real world adversary can lead to rejecting protocols which are secure enough in practice, but susceptible to exotic attacks in the model. Note that security is in many situations achieved at the expense of efficiency.

The intruder model which is most often used in the security literature is called the *Dolev-Yao* intruder model after [DY81, DY83]. In this model, there is one intruder, comprising all the outsider and insider corrupted parties, which has control over the entire communication network. It intercepts all messages that have been transmitted and can store them for its future use. It can encrypt, decrypt and sign messages if it knows the corresponding keys; it can compose and send new messages using its knowledge as well. It can remove or delay messages in favour of others being communicated. "[It] is a legitimate user of the network, and thus in particular can initiate a conversation with other users" [DY83]. A formal definition of this intruder model is given in § 4.

As the Dolev-Yao intruder can destroy all the transmitted messages, no liveness property can in general be proved in this model. Achieving liveness in distributed systems requires some guarantees on the quality of the communication media. This is because disrupting the communication channels can potentially isolate the parties involved in the system and, then, reaching any non-trivial common state simply becomes impossible, cf. [FLM86]. The Dolev-Yao intruder model is thus too strong for verifying liveness, and its power regarding destroying messages has to be limited. But, how much limited? In practice, protocols rely on *resilient* channels. According to Asokan "a message inserted into a resilient channel will eventually be delivered" [Aso98]. Note that only an eventual delivery is sufficient to satisfy resilience and, in particular, no conditions are put on delivery time. It is in fact not hard to realise a resilient channel, although the resulting channel might not be suitable for fast

communications. For instance, if Bob blocks Alice's Internet connection at home to prevent her from accessing her electronic bank account, Alice can always walk to a nearby branch of the bank. Protecting Alice from intruders who can physically isolate her falls beyond what security protocols provide.

The first topic that we address in § 5 concerns formalising an intruder model which does not indefinitely delay delivering messages. We introduce a carefully crafted *fairness constraint* that excludes, from the model, the executions in which messages are indefinitely delayed. A fairness constraint, not to be confused with fairness in exchange, refers to the *scheduler* in the model. Security protocols are modelled as a collection of independent processes which interact by message passing. In the model, conceptually, a scheduler assigns turns to these processes to send or receive message or perform internal computations. A fairness constraint, intuitively, determines which schedules are not realistic. For instance, if Alice is never scheduled in the model, no security protocol can authenticate her to Bob, simply because Alice is never given a chance to take any step in the model. Such unrealistic executions need to be omitted from further (formal) analysis.

The fairness constraint which reflects resilient channels turns out to be complicated, mainly because it depends on the events that have occurred in the past. Simplifying this fairness constraint in a way which suits automatic formal verification techniques is the second subject studied in § 5. We achieve this by slightly modifying the Dolev-Yao intruder. Roughly speaking, the modified Dolev-Yao intruder marks certain messages with a special tag. These tags are not visible to the honest parties which interact with the intruder, and are used solely in the formalisation. The resulting model, i.e. the modified intruder plus the simple fairness constraint, is proved to be equal to the Dolev-Yao intruder model under the resilient communication channels assumption. Several examples are presented to motivate and clarify the proposed intruder model, which is suitable for efficient verification of liveness properties of security protocols.

In § 6, we use the intruder model and fairness constraint developed in § 5 to formally verify two protocols: A fair payment protocol for purchasing *time sensitive* data in mobile environments [VPG01], and a fair digital rights management scheme [NPG+05, TKJ07].

Time sensitive information, such as current stock exchange quotes and location dependent information in mobile services, may lose their value over time. To enable the customers to roll back the exchanges which are excessively delayed, the protocol of [VPG01] assumes that each customer is equipped with a smart card. The protocol is intended for mobile applications, where "a vendor sells a digital [time sensitive item] to a mobile customer who pays for it electronically" [VPG01]. We observe that this payment protocol, analysed in § 6, does not achieve one of its liveness goals, namely the termination for the vendor. As our intruder model is geared towards verifying liveness, detecting this flaw serves as an empirical basis for the effectiveness of the model.

The fair digital rights management scheme of [NPG+05, TKJ07] aims at providing a secure environment for secure exchange of *content-right* bundles among trusted computing

devices, such as iPods. Trusted devices are tamper-resistant hardware that follow their certified software. They use (e.g. render) each *content* exactly as is instructed by its associated *right*. A legitimate content provider, such as iTunes, is the original distributor of protected content-right bundles. A content purchased from the provider can further be traded among the trusted devices, if this is allowed by the associated right. This scheme heavily relies on trusted devices. We describe how our formalisation reflects the characteristics of these devices.

The last chapter of the thesis is devoted to a *partial order reduction* technique for verifying optimistic fair exchange protocols. In automatic verification techniques (such as model checking and constraint solving), we often require to enumerate all possible interleavings of actions performed by protocol participants. Partial order reduction algorithms identify and avoid generating identical interleavings, modulo the properties that are to be verified, to reduce the time and memory used in verification. A characteristic of optimistic fair exchange protocols is that their participants have certain choice points in the course of the protocol. The partial order techniques previously developed for security protocols (such as [CJM00a]) do not consider such choice points and, thus, are not readily applicable to these protocols. In § 7, we extend the reduction technique of [CJM00a] to so-called branching security protocols, in which participants may have choice points.

Origins of the chapters

Section 2.2.1 is based on an unpublished literature review that Simona Orzan and me have recently conducted on the solvability of fair exchange.

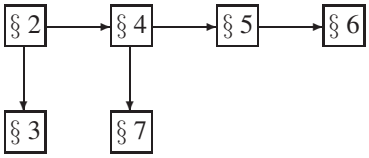
Chapter 3 is based on [CTM07]. Jan Cederquist, Ricardo Corin and me initially discussed the possibility of using forward key chains to reduce the amount of information stored by trusted entities in the non-repudiation protocol of [CCT05]. Jan and me later on followed the idea and designed a certified email protocol based on key chains. Sjouke Mauw helped with the cryptographic details of this protocol.

Chapter 5 is mainly based on [CT06]. Back in 2004, while working on [CT04], we noticed the difficulty of adding resilient channels to the Dolev-Yao intruder model in a provably correct manner. We published an early technical report on this issue [CT05], and in [CT06] we give an intruder model which is equivalent to the Dolev-Yao intruder, except that it cannot indefinitely delay messages over resilient channels. This model was first used in [CCT05] to verify a fair non-repudiation protocol. The case studies reported in chapter 6 are based on [CT04, CCT05, TKJ07].

Chapter 7 describes a partial order reduction algorithm for security protocols. This algorithm has been presented in [FTW07]. The experimental results are taken from [TWL07].

Road map

The thesis is written in separate chapters. Each chapter comes with its own road map and short introduction. Related and future work, when relevant, and conclusions appear for each chapter separately. These chapters are however not entirely self contained. Some preliminary notions and notations are given in § 2 and § 4, and are only referred to in other chapters. Therefore, the reader is encouraged to read § 2 before § 3, and chapters 2 and 4 before §§ 5 – 7. The following diagram shows the dependency relation between the chapters of the thesis.



Chapter 2

Fair exchange

Road map In this chapter we first introduce security protocols in general, and then focus on fair exchange protocols. Next, we review some of the main ideas and results on tackling the fair exchange problem

2.1 Security protocols

A distributed system consists of a finite number of processes that interact by some communication means. In the thesis, we only consider the message passing setting, where processes communicate by sending and receiving messages over communication channels. A collection of communication channels is called a communication network. We write

$$A \rightarrow B : m,$$

when process A submits message m to the communication network, with the intention that it should be delivered to process B. A *synchronous* channel guarantees to deliver messages in a timely manner, with a pre-known time bound, while *asynchronous* channels deliver messages eventually, but no time bounds are put on them. Channels may in general lose or duplicate messages. Unless explicitly stated, we do not consider such faulty channels. [1]

A *protocol* assigns an algorithm to each participating process, such that using the communication primitives available to the processes, they can achieve a certain common goal. A synchronous protocol assumes that the processes execute in lock-step, i.e. there is an upper bound on the difference between computation speeds of each two processes, and that the communication network is synchronous. Asynchronous protocols do however not assume these properties. A *fault tolerant* protocol achieves its goal even if some of the participating processes are faulty.

Different failure models are used in distributed systems to characterise how a faulty process may misbehave. One of the simplest model is *crash* failure, in which the failed process simply dies, i.e. ceases to act afterwards. It may however send out any subset of the messages that it was supposed to send at the moment of crash. In the *Byzantine* failure model [LSP82], a faulty process may deviate from the algorithm assigned to it in any fashion it wants, but its

[1] As is described later, the attacker is modelled as a process that can inject messages into the channels and remove messages from (some of) the channels, even if the channels are assumed to be non-faulty.

view is local, i.e. it only sees what is passed to it by its neighbours, and its effect is local as well, i.e. it can only send messages to its neighbour processes.

Cryptographic or *security* protocols are fault tolerant protocols which use cryptography to attain their goals. In computer security, the Dolev-Yao model [DY81, DY83], denoted $\mathcal{DY}$, is usually considered as the hostile environment model. In this model, there is one malicious process (called attacker, intruder, etc), comprising all the outsider and insider corrupted parties, which has control over the entire communication network. [2] It intercepts all messages that have been transmitted and can store them in its knowledge set. It can also remove or delay messages in favour of others being communicated. "[It] is a legitimate user of the network, and thus in particular can initiate a conversation with other users" [DY83]. Security protocols are typically designed to protect the interests of the honest participants, i.e. those who faithfully follow the protocol, in presence of the $\mathcal{DY}$ attacker (for a critique on the $\mathcal{DY}$ model in face of the emerging mobile ad-hoc protocols see, e.g., [Gli07]). Honest participants only follow the protocol, and, are in general not required to take any steps to detect or thwart attacks.

The $\mathcal{DY}$ attacker can be seen as a Byzantine process which is sitting in the centre of a star-like network topology. All other processes therefore communicate through $\mathcal{DY}$, hence the network being of connectivity 1. [3] Network connectivity indeed plays a role in the possibility of distributed tasks, performed in presence of malicious parties, see [FLM86, Syv97].

2.1.1 Ideal cryptography

The $\mathcal{DY}$ model is usually associated with the *ideal cryptography* assumption. In ideal cryptography, messages exchanged in protocols are thought of as formal terms rather than strings of bits [DY81, DY83]. Cryptographic apparatus available to the participants are therefore abstract operations which can manipulate messages solely according to the formal rules which define them. We start describing this concept with an example. Let $\{m\}_k^s$ denote encrypting message m with key k using a symmetric encryption algorithm (hence the superscript s). In ideal cryptography, without knowing k, no information on m can be extracted from $\{m\}_k^s$ and, moreover, k cannot be inferred from $\{m\}_k^s$ alone.

Ideal cryptography thus adopts a black and white view of the matter, while cryptography traditionally relies on probabilistic reasoning and computational tractability, see e.g. [MVO96, Gol05]. This simplification allows us to abstract away the cryptographic functions used in a protocol and, instead, focus on the security of the protocol itself (see example 2.1 below). There are however drawbacks. For instance, consider a protocol which requires Alice to send message $\{0\}_k^s$ to Bob. If the *exclusive or* function, denoted $\oplus$, is used for encryption, i.e. $\{m\}_k^s = m \oplus k$, then ideal cryptography does not provide a sound abstraction for the imple-

[2] Any number of $\mathcal{DY}$ attackers can be modelled as a single $\mathcal{DY}$ attacker by merging their knowledge sets [SM00].

[3] A network has connectivity c iff at least c nodes need to be removed to disconnect the network.

mentation. This is because an attacker who knows the terms of the protocol [4] can freely derive k, as $0 \oplus k = k$, while this is deemed impossible according to ideal cryptography. The ideal cryptography assumption is therefore of little use when reasoning about protocols which inherently rely on algebraic properties of encryption algorithms (for instance, Diffie-Hellman's key exchange protocol [DH76]).

Below, we formally define the syntax of messages exchanged in security protocols. Let $\mathcal{T}$ be a set of atomic terms, and $\mathcal{P} \subseteq \mathcal{T}$ be the set of protocol participants' identities. It is assumed that $\mathcal{P}$ is publicly known.

2.1. DEFINITION. *The syntax of messages is formally defined as:*

$$m ::= t \mid \ m, m \mid h(m) \mid \{m\}^s_m \mid \{m\}^a_{pk(p)} \mid \{m\}^a_{sk(p)} \ ,$$

where $t \in \mathcal{T}$ represents any atomic term, and $p \in \mathcal{P}$.

Intuitively, $\cdot, \cdot$ denotes pairing and $h(\cdot)$ represents a one-way hash function. The term $\{m_1\}^s_{m_2}$ is the encryption of m_1 with key m_2 using a symmetric encryption algorithm.

We differentiate symmetric and asymmetric encryption techniques.[5] For $p \in \mathcal{P}$, $pk(p)$ and $sk(p)$ denote, respectively, the public and secret private keys of $p \in \mathcal{P}$. Encrypting m with p's public key using an asymmetric encryption algorithm is denoted by $\{m\}^a_{pk(p)}$. Encrypting with p's secret private key $\{m\}^a_{sk(p)}$ represents signing. When public key encryption is used, usually, a public directory associates participants with their corresponding public keys. Following [DY83], we assume "the public directory is secure and cannot be tampered with; everyone has access to all $[pk(p)]$; only $[p]$ knows $[sk(p)]$".

The attacker's deduction capabilities in the ideal cryptography world of [DY83] can informally be described as follows:

- Pairing: m_1, m_2 can be composed by knowing both m_1 and m_2. Conversely, from (m_1, m_2), both m_1 and m_2 can be extracted.
- Hash functions: $h(m)$ can be composed by knowing m; and from $h(m)$ alone, m cannot be extracted.
- Symmetric encryption: $\{m\}^s_k$ is incomprehensible without knowing k. Similarly, without knowing m and k, $\{m\}^s_k$ cannot be constructed, except by eavesdropping.
- Asymmetric encryption: Recall that $pk(p)$, for all $p \in \mathcal{P}$, is known to everyone. Therefore, anyone can construct $\{m\}^a_{pk(p)}$ when knowing m. However, $\{m\}^a_{pk(p)}$ reveals nothing about m, without knowing $sk(p)$. Similarly, without knowing m and $sk(p)$, $\{m\}^a_{sk(p)}$ cannot be constructed, except by eavesdropping.

[4] Following Kerckhoffs's principle [Ker83], details of cryptographic algorithms and protocols must not require secrecy, i.e. they are assumed to be publicly known. The keys used in the algorithms may however require secrecy.

[5] In symmetric encryption both encryption and decryption keys are assumed to be secret, while in asymmetric (or public key) encryption, the encryption key is publicly known. Thus, it can be assumed that in symmetric encryption, encryption and decryption keys are equal.

As cipher algorithms are assumed to be ideal in the $\mathcal{DY}$ model, the attacker can subvert a protocol only by exploiting the *logic* of the protocol. To clarify the idea, we present an example from Dolev and Yao's seminal paper [DY81].

2.1. EXAMPLE. *The problem is "transmitting a secret plain-text M between two users". Consider the following protocol addressing this problem:*

$$A \to B:\ \{M, A\}^a_{pk(B)}$$
$$B \to A:\ \{M, B\}^a_{pk(A)}$$

Dolev and Yao prove that, in their model, this protocol does not reveal M to the attacker, i.e. A can consider M a secret. However, "improving" the protocol by adding another layer of encryption, as shown below, makes the protocol breakable.

$$A \to B:\ \{\{M\}^a_{pk(B)}, A\}^a_{pk(B)}$$
$$B \to A:\ \{\{M\}^a_{pk(A)}, B\}^a_{pk(A)}$$

In this protocol, B, the responder, peels off the message which is encrypted for him and sends it back to A, the originator. Intuitively, B acts as an oracle [6] *that decrypts the messages which are encrypted using $pk(B)$. The $\mathcal{DY}$ attacker Z can thus get access to M:*

$$A \to B:\ \{\{M\}^a_{pk(B)}, A\}^a_{pk(B)}$$
$$B \to A:\ \{\{M\}^a_{pk(A)}, B\}^a_{pk(A)}$$
$$Z \to A:\ \{\{\{M\}^a_{pk(A)}, B\}^a_{pk(A)}, Z\}^a_{pk(A)}$$
$$A \to Z:\ \{\{\{M\}^a_{pk(A)}, B\}^a_{pk(Z)}, A\}^a_{pk(Z)}$$
$$Z \to A:\ \{\{M\}^a_{pk(A)}, Z\}^a_{pk(A)}$$
$$A \to Z:\ \{\{M\}^a_{pk(Z)}, A\}^a_{pk(Z)}$$

This attack shows a flaw in the protocol's logic, which persists even if the cryptographic primitives used are secure.

We are now ready to formally define the $\mathcal{DY}$ attacker's deduction abilities. Let Γ be a set of messages. We write $\Gamma \vdash m$ to denote that the $\mathcal{DY}$ attacker can construct m using Γ.

2.2. DEFINITION. *Below, the rules labelled with $\mathcal{CR}$ and $\mathcal{DR}$, respectively, represent composition and decomposition abilities of the $\mathcal{DY}$ attacker. For a set of messages Γ we have:*

- *Members of Γ are readily available to $\mathcal{DY}$.*

$$\frac{m \in \Gamma}{\Gamma \vdash m}\ \mathcal{CR},\mathcal{DR}$$

[6] According to [Car94], Dolev and Yao were the first to present examples of oracle flaws in security protocols. These flaws are not limited to pedagogical examples. See, e.g., [NTCT07], where an oracle flaw in a recent *assured delete* protocol is reported.

- *$\mathcal{DY}$ can pair messages and can decompose pairs.*

$$\frac{\Gamma \vdash m_1 \qquad \Gamma \vdash m_2}{\Gamma \vdash m_1, m_2}\,\mathcal{CR} \qquad \frac{\Gamma \vdash m_1, m_2}{\Gamma \vdash m_1}\,\mathcal{DR} \qquad \frac{\Gamma \vdash m_1, m_2}{\Gamma \vdash m_2}\,\mathcal{DR}$$

- *$\mathcal{DY}$ can hash messages.*

$$\frac{\Gamma \vdash m}{\Gamma \vdash h(m)}\,\mathcal{CR}$$

- *$\mathcal{DY}$ can perform symmetric encryption and decryption.*

$$\frac{\Gamma \vdash m_1 \qquad \Gamma \vdash m_2}{\Gamma \vdash \{m_1\}^s_{m_2}}\,\mathcal{CR} \qquad \frac{\Gamma \vdash \{m_1\}^s_{m_2} \qquad \Gamma \vdash m_2}{\Gamma \vdash m_1}\,\mathcal{DR}$$

- *$\mathcal{DY}$ can perform asymmetric encryption and decryption.* [7]

$$\frac{\Gamma \vdash m}{\Gamma \vdash \{m\}^a_{pk(p)}}\,\mathcal{CR} \qquad \frac{\Gamma \vdash m \qquad \Gamma \vdash sk(p)}{\Gamma \vdash \{m\}^a_{sk(p)}}\,\mathcal{CR}$$

$$\frac{\Gamma \vdash \{m\}^a_{pk(p)} \qquad \Gamma \vdash sk(p)}{\Gamma \vdash m}\,\mathcal{DR}$$

Decidability issues

Given a non-empty set of messages Γ, the set $\{m \mid \Gamma \vdash m\}$, though infinite, is recursive, i.e. for an arbitrary message m, whether $\Gamma \vdash m$ holds or not, is decidable [CJM98, CLS03]. The question $\Gamma \vdash m?$, also known as *ground reachability*, intuitively characterises what a passive attacker can learn by eavesdropping, when Γ represents the collection of messages exchanged in the protocol. More interesting is the problem of *reachability* in cryptographic protocols: Given a protocol description, to determine whether the $\mathcal{DY}$ attacker, possibly playing an active role, can use the protocol to reach an *error* state, e.g. disclosure of a secret.

Solvability of the reachability problem depends not only on the attacker's power, but also on the participant's computing capabilities. Recall that in § 2.1 we declared that a protocol "assigns an algorithm to each participating process". Using Rice's theorem [Ric53] it comes

[7] Note that $\mathcal{DY}$ is not allowed to extract message contents from signatures alone. This corresponds to assuming that signature schemes do not provide message recovery. Any signature scheme with message recovery (such as [RSA78]) can nonetheless be converted into a signature scheme which conceals message contents by simply signing hash values of messages. See § 11.2.3 of [MVO96] for detailed discussions.

as obvious that, with this liberal definition of participants, the reachability problem is undecidable, even if no cryptographic primitives are used, cf. [HT96]. The set of secure protocols in this setting is in fact not recursively enumerable (see, e.g., [Tor03]).

Putting certain restrictions on the participants can fortunately make the reachability problem decidable. Even and Goldreich [EG83] study protocols in which participants can take a finite number of steps, and in each step can only apply (or cancel) encryption and appending on a received message and send the result back to the network, hence being called *ping-pong* protocols. [8] The reachability problem is decidable for ping-pong protocols even when the participants can replicate (i.e. each participant can engage in an unbounded number of protocol executions) [EG83]. However, if the participants are allowed to apply these operations on half-words (i.e. parts of messages) the reachability problem becomes undecidable [EG83] (see also [ALV01]). In fact, if the participants of ping-pong protocols are allowed to have cyclic specifications, the reachability problem becomes undecidable, even when no half-word computations or process replications are allowed [HS04].

On the positive side, the reachability problem is decidable, when the participants are acyclic, cannot replicate and are specified as simple rewrite rules or process terms [ALV01, Bor01, RT01, MS01]. Roughly, the idea is to model the infinitely branching behaviour of $\mathcal{DY}$ using symbolic transitions, the analysis of which turns out to be amenable to automation. See [CLS02, TEB06, CDL06] for general surveys on decidability results in the $\mathcal{DY}$ model.

Decidability results encourage using ideal cryptography, as automatic security proofs are thereby facilitated. Automation however comes at the expense of sacrificing soundness: A protocol proved secure in the ideal cryptography model can be susceptible to attacks which exploit algebraic properties of cryptographic apparatus used in instantiating the protocol. For example, see Ryan and Schneider's attack [RS98b] on a protocol which, assuming ideal cryptography, had been proved correct in [Pau97]. Proving the actual software or hardware implementation of a protocol secure is yet another (perhaps big) step farther.

To lift $\mathcal{DY}$'s ideal cryptography limitations, two major approaches have been investigated: First, is to enrich the $\mathcal{DY}$ model with the features of cryptographic primitives used in implementing security protocols. For instance, as $\oplus$ is abundantly used in cryptographic protocols, we could add the following rule to the list of $\mathcal{DY}$'s deduction rules:

$$\frac{\Gamma \vdash m_1 \qquad \Gamma \vdash m_2}{\Gamma \vdash m_1 \oplus m_2}\, \mathcal{CR}$$

Defining the corresponding decomposition rule, i.e. to specify what can be derived from $m_1 \oplus m_2$, is not obvious. For example, the straightforward rules $(\Gamma \vdash m_1 \oplus m_2,\ \Gamma \vdash m_2) \implies \Gamma \vdash m_1$ and $(\Gamma \vdash m_1 \oplus m_2,\ \Gamma \vdash m_1) \implies \Gamma \vdash m_2$ are not sufficient. This is because $\{a \oplus b \oplus c, a \oplus c\} \vdash b$, which indeed holds, does not follow from these rules, if commutativity of $\oplus$ is not taken into account. For more on this see [CLS03, CKRT03].

[8] Ping-pong protocols were first studied by Dolev and Yao [DY81].

The second approach to reconcile the ideal cryptography assumption with the actual cryptographic primitives is to determine the criteria under which protocols that are proved correct in the $\mathcal{DY}$ model may be securely instantiated with real cryptographic primitives. For more on this approach see, e.g., [AR00, BP05].

2.2 Fair exchange protocols

Fair exchange protocols are a class of security protocols which aim at exchanging digital items in a *fair* manner. Informally, fair means that all involved parties receive a desired item in exchange for their own, or none of them does so. There exist various flavours of the fair exchange (FE) problem, e.g. fair contract signing (CS), fair payment (FP), fair certified email (CEM), fair exchange of secrets (ES) and fair non-repudiation (NR) protocols. Below, we introduce these FE variants via examples:

- CS: Alice and Bob have agreed on a contract and would like to sign it electronically. [9] Alice gives her signature on the contract to Bob, only if she receives the contract signed by Bob. Similarly, Bob signs the contract and passes it to Alice, only if he receives Alice's signature. In short, they want to *simultaneously* exchange their signatures.

- FP: Alice sees Bob's electronic book on the Internet and wants to buy it, but she does not want to send her digital coins to Bob before receiving the book and making sure that it is indeed what he has advertised. Similarly, Bob does not want to send his electronic book to Alice before receiving Alice's coins and making sure that they are genuine. They want to simultaneously exchange their digital items.

- CEM and NR: Alice wants to send an email to Bob in exchange for a receipt. The receipt is a proof that shows Bob has received the email. It thus has to in some way specify the content of the email. Bob is in turn willing to send back the receipt to Alice only if he actually receives Alice's email. Notice that in this case, Alice and Bob do not aim at simultaneous exchange. This is because of the inherent asymmetry of the problem: The receipt depends on the content of the email.

- ES: Alice and Bob each possess a secret that is not known to the other one. Alice would like to exchange her secret with Bob's in a "secret exchange party", but she does not want to reveal her secret unilaterally, and likewise for Bob. Note that this exchange is meaningful only if Alice and Bob know "something" about the other party's secret. Otherwise, any protocol that distributes random bits would be acceptable, since Alice would think that the junk is actually Bob's secret, and similarly for Bob.

[9] Fair and private contract *negotiation* protocols are discussed in, e.g., [FA05].

Although these problems are similar and we refer to them collectively as FE, there are subtle differences between them. For instance, CEM and CS are different in simultaneity, and CEM is different from ES in that the receipt of an email is not precisely defined in CEM, and can thus be different from one protocol to another (as it happens to be in practice), while ES is to exchange the secrets themselves. It is also notable that in ES the participants are assumed to know something about the other party's secret, which is a trivial precondition when it comes to CS: Signatures are the subject of exchange, and digital signatures always have a verification algorithm associated to them. In NR protocols, the aim is to fairly exchange evidences, such that Alice receives an evidence of receipt iff Bob receives an evidence of origin on a certain document. Moreover, the participants are required to be accountable for (i.e. cannot deny) the promises they utter in the course of the exchange. We do not distinguish NR and CEM protocols in this document, since these are conceptually very similar. The challenge in NR protocols is to exchange the evidences in a fair way, otherwise, non-repudiation of the evidences can easily be achieved using standard digital signatures, cf. [ZG97b].

In the literature, there is no consensus on what FE protocols (or its variants) have to provide. Below, we informally describe the goals that a generic FE protocol for two parties, named A and B below, achieves (à la [Aso98]).

- *Timeliness* states that any honest participant can terminate the exchange unilaterally, i.e. without any help from the opponent. Timeliness guarantees that none of the participants can arbitrarily force the other one to wait for the termination of the exchange.
- *Effectiveness* states that if both parties are honest and willing to perform the exchange and none of them abandons the exchange, then the protocol terminates in a state where A has B's item and vice versa. This is in fact a functional sanity check for the protocol.
- *Fairness* states that if A terminates the protocol in a state where A has B's item, then B terminates the protocol in a state where B has A's item, and vice versa. This property is often referred to as *strong* fairness [Aso98].

Any protocol that achieves these goals is said to solve FE. We remark that each variant of FE can have its own specific requirements. See [BVV84] for a formal study on the relations between some of the FE variants.

2.2.1 Solvability of fair exchange

In this section, we focus on solvability of the FE problem.

Synchronous protocols

Even and Yacobi [EY80], and independently Rabin [Rab81], studied simple variants of the FE problem. In [EY80], a notion of mutual signature on a message (the CS problem) is studied.

They informally reason that "if the judicator is not active during the ordinary operation of the system", then no two-party protocol can achieve *agreement*, where agreement means that when a party can compute the signature, the other one can also do so. Their argument goes as: "Assume that, after n communications, [Alice] has sufficient information for efficient calculation of [the mutual signature], but that this is not true for $n-1$ communications. We conclude that [Bob] transmits the n^{th} communication, and therefore the first time [Bob] has sufficient information is after n' communications, where $n' \neq n$. This contradicts [the definition of agreement]".

Rabin considers the similar problem of simultaneous exchange of secrets between two non-trusting entities Alice and Bob (the ES problem). He deduces that the problem is unsolvable: "Any [exchange] protocol must have the form: Alice gives to Bob some information I_1, Bob gives to Alice J_1, Alice gives to Bob I_2, etc. There must exist a first k such that, say, Bob can determine [Alice's secret] from $I_1, \ldots, I_k$, while Alice still cannot determine [Bob's secret] from $J_1, \ldots, J_{k-1}$. Bob can withhold J_k from Alice and thus obtain [Alice's secret] without revealing [his own secret]".

Since these problems are instances of FE, their unsolvability implies unsolvability of FE in the corresponding models. Both these arguments clearly stress on the malicious act of withholding the last message. They can thus be summarised as: No two-party protocol with one Byzantine process, even with synchronous communication channels, can solve FE. This result naturally carries over to asynchronous protocols. We remark that a crucial feature of this model is that no party is *trusted* by other process(es). A process is trusted iff it is publicly known that the process is (and remains) non-faulty. DeMilo, Lynch and Merritt formalised the impossibility arguments mentioned above in [DLM82].

In [BGW88] and, independently, in [CCD88], the authors derive general solvability results regarding the secure multi-party computation (SMPC) problem. These results are pertinent to our discussion, since FE appears to be an instance of SMPC. In [BGW88] it is established that, in a fully connected network of synchronous channels, n-party SMPC, and thus FE, is achievable if there are at most t Byzantine participating processes, with $t < \frac{n}{3}$. They also prove that there exist SMPC problems which, with $t \geq \frac{n}{3}$ Byzantine processes, are unsolvable for n parties. The results of [EY80, Rab81] clearly show that FE is one of these problems. See [GL02a] for an excellent review on further developments in SMPC.

We note that the possibility results of [BGW88, CCD88] do *not* imply the solvability of FE in the $\mathcal{DY}$ model, simply because the connectivity of the network is 1 in the $\mathcal{DY}$ model, while these results are stated in complete graph topologies. In fact, reaching *distributed consensus*, a problem conceptually similar to FE, is impossible if the network connectivity is less than $2t+1$, with t Byzantine processes [FLM86].

Asynchronous protocols

In asynchronous systems, the impossibility result of [FLP85] and its extension in [MW87] imply that multi-party FE is unsolvable even if one of the processes is subject to crash failure. [10] For two-party exchanges, this result has been derived in [PG99] by reducing the distributed consensus problem to FE. It is worth mentioning that the impossibility result of [EY80] (and [Rab81, DLM82]) is based on the malicious act of withholding parts of information, whereas [PG99] proves impossibility of FE in the presence of benign, but not "malicious", failures, as a result of lack of knowledge to decide termination in asynchronous systems. These, thus, concern orthogonal difficulties in solving FE, and none of them directly implies the other one.

Up until now, we focused on the effects of process failures, as opposed to channel failures, on solving the FE problem. Below, we consider the case of lossy channels, while assuming that processes are all honest (i.e. correct). In distributed computing, the limitations on reaching agreements in the presence of lossy channels is usually described using the *generals paradox* [Gra78]: "There are two generals on campaign. They have an objective (a hill) that they want to capture. If they simultaneously march on the objective they are assured of success. If only one marches, he will be annihilated. The generals are encamped only a short distance apart, but due to technical difficulties, they can communicate only via runners. These messengers have a flaw, every time they venture out of camp they stand some chance of getting lost (they are not very smart.) The problem is to find some protocol that allows the generals to march together even though some messengers get lost."

Gray informally shows that such a protocol does not exist [Gra78]. This has later on been formally proved in, e.g., [YC79, HM84]. Note that the generals problem can be reduced to two-party FE in a straightforward way. The impossibility result stated above, thus, implies that FE is unsolvable in the presence of channel failures, even with honest participants.

Halpern and Moses furthermore prove that in the presence of channel failures, "any protocol that guarantees that whenever either party attacks the other party will *eventually* attack, is a protocol in which necessarily neither party attacks" [HM84]. This result implies that in optimistic FE protocols even with all honest participants, resilient channels are unavoidable (optimistic protocols are described below).

2.2.2 A selective literature review

> The hare said, "We need a just judge to hear us both, and based on fairness, settle the dispute". The partridge said, "On the shores of this river lives a pious cat who fasts all day and prays all night".
>
> Nasrallah Monshi's *Kalileh o Demneh* (ca. 1200 AD)

[10] We note that the possibility results of [BCG93] are not in contradiction to this impossibility statement, as [BCG93] only provides probabilistic termination (or timeliness).

Below, we review some of the main ideas and results on solving the FE problem in the $\mathcal{DY}$ model. This review is selective. In particular, we do not touch upon various multi-party and valued-added FE protocols (some of these are however discussed in subsequent sections). Synchronous protocols are also mostly absent from our review. For general surveys on the topic see also [Aso98, Sch00, KMZ02, Cha03, Kre03, Nen05, Gon05].

There are three general constructions for FE, based on the degree of the involvement of trusted third parties (TTP). The first group needs no TTPs, e.g. the protocols of [Blu81, Rab81, EGL85, Cle90, BOGMR90, MR99], see also [FGY92] for a historical survey on these protocols. These are based on gradual release of information or gradual increase of privileges and require exchanging many messages to approximate fair exchange, as deterministic asynchronous FE with no trusted parties is impossible (see § 2.2.1). Protocols of the second group need the TTP's intervention in each exchange, e.g. see [BT94, CTS95, ZG96a, ZG96b, DGLW96, FR97, AG02]. In the literature, these are sometimes called protocols with *in-line* or *on-line* TTPs. [11] These protocols have a fixed, usually small, number of message exchanges, and are thus more appealing in practice. However, the TTP can easily become a communication bottleneck or a single target of attacks, as it is involved in each exchange. Protocols of [Rab83, RS98a] can also be listed in the second group as they require the TTP to be active during each exchange. However, a slight difference is that, intuitively, the TTPs in the latter protocols need not be aware of being involved in such exchanges. The third group of FE protocols, known as *optimistic* protocols, require the TTP's intervention only if failures, accidentally or maliciously, occur, e.g. see [Eve83, ZG97a, ASW97, Mic97, ASW98a, ASW98b, BDM98, ZDB99, MK01, Mic03, PCS03, DR03, CCT05]. Therefore, honest parties that are willing to exchange their items can do so without involving any TTP. Optimistic protocols are called protocols with *off-line* TTPs.

Optimistic FE protocols typically consist of three sub-protocols: *exchange protocol* (also called *optimistic protocol*), *recovery protocol* and *abort protocol*. Figure 2.1 depicts a generic optimistic FE protocol. The regions in which recovery and abort protocols are alternative possibilities are also shown in the figure. In the exchange protocol, that does not involve the TTP, the agents first *commit* to release their items and then they actually release them. The commitments and the exchanged items are respectively denoted by c_A, c_B and i_A, i_B in figure 2.1. Process A can run the recovery protocol if the opponent B has committed to exchange, but A has not received B's item, and vice versa. A participant aborts (cancels) the exchange if she does not receive the opponent's commitment to the exchange. Optimistic protocols typically require the communication channels to and from the TTP to be *resilient*, i.e. messages be delivered within an arbitrary but finite amount of time. [12] This guarantees

[11] On-line TTPs, although being involved in each exchange, act only as a light-weight notary, as opposed to to in-line TTPs which directly handle the items subject to exchange, cf. [ZG96c].

[12] Note that resilient channels are required in the $\mathcal{DY}$ model, where there is no direct (secure) link between protocol participants, i.e. messages pass through the attacker process. Therefore, in contrast to asynchronous channels which guarantee eventual delivery from one end of the channels to the other, resilience limits the attacker's abilities.

that, in case of failure, protocol participants can eventually consult the TTP. Assuming that malicious or accidental failures are infrequent (hence being optimistic), optimistic protocols put less load on the TTP, compared to the protocols of the second group.

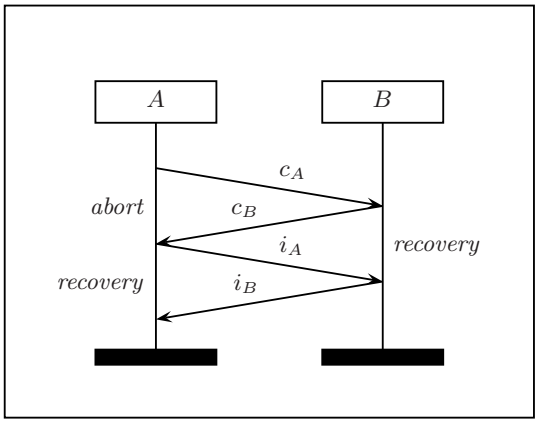


Figure 2.1: *Optimistic FE - exchange pattern*

Pivotal to the working of optimistic protocols is the nature of the items that are subject to exchange. It has been shown in [SW02] that optimistic FE is impossible if the exchanged items are neither *generatable* nor *revocable*. [13] An item is generatable if the TTP can generate the item from a participant's commitment to release that item, and an item is revocable if the TTP can revoke the validity of that item. In general, digital items are neither generatable nor revocable. However, cryptographic tools, such as verifiable encryption, can make certain digital items generatable. For instance, see [ASW98b, Che98, PCS03, DR03, Ate04, DJH07] for techniques to enable the TTP to generate participants' signatures from their commitments (see also [RR00]). We remark that these techniques involve heavy cryptographic machinery. In contrast, there are not many digital items that can be revoked by the TTP (see below).

The impossibility result of [SW02] comes as no surprise when noticing that if a wronged Bob resorts to the TTP, he wishes (at least) one of the following services: Either the TTP can generate the item that he has expected, which is impossible if the item is not generatable, or the TTP can revoke the item that he has lost (i.e. currently being in the possession of Alice), which is impossible if the item is not revocable. The TTP can however provide Bob with an affidavit declaring that Bob has indeed been cheated (by Alice). In this case, Bob only achieves *weak* fairness [Aso98], which might not satisfy Bob. Below, we explore how such affidavits can be used to provide strong fairness in CS, CEM and NR protocols.

The goal is to provide strong fairness without using costly cryptographic tools such as verifiable encrypted signatures. The idea is to exploit a freedom that is inherent to the def-

[13] In general, no such restriction applies to FE protocols with in-line or on-line TTPs.

initions of CS, CEM and NR: In these FE variants, the protocol (designer) is free to define what constitutes a, e.g., mutually signed contract, a signed receipt, or evidence of origin. Therefore, these protocols devise dispute resolution procedures to evaluate (or interpret) the digital assets that are collected in the protocol. Dispute resolution procedures can thereby be tailored to grade affidavits from the TTP as, for instance, a valid evidence of origin. This idea has been used in many FE protocols such as [ZG97a, ASW98a, ZDB99, KMZ02, GRV05, CCT05, CTM07]. Note that these protocols enforce the structure of the exchanged items, hence being called *invasive* [ASW98a]. Non-invasive protocols are more favourable, but come at high computation costs, as they rely on unconventional cryptographic tools, as in, e.g., the signature exchange protocols of [ASW98b].

A partial remedy to invasiveness is to make the TTP *invisible* [Mic97], such that there would not be any difference between the evidences collected in optimistic runs and those issued by the TTP. Note that the structure of the evidences is still determined by the protocol, hence the result may be an invasive protocol (e.g. as in [Mic03]). The exchanged items however would not reveal whether the TTP was involved in the exchange or not. For protocols with invisible, or *transparent*, TTPs see, e.g., [Mic97, ASW98a, MK01, MS02, Mic03, Ate04]. As is put by Asokan, "typically, non-invasiveness implies invisibility of third party" [Aso98].

Now we turn to fair exchange of revocable items. Generally, it is hard to revoke digital items. However, certain payment systems can in principle provide revocable coins, e.g. see [JY96, Vog03]. Fair payment protocols which employ revocable money (orders) are presented in [ASW98a, Vog03]. A separate group of protocols for exchanging revocable items exploits the freedom in the definition of CS, CEM and NR, just as mentioned earlier. These not only prescribe a tailored dispute resolution procedure to grade the TTP's messages as valuable evidences, but also they require the TTP to in some situations participate in the dispute resolution phase of the protocol in order to revoke evidences collected by the participants. Examples of protocols following this idea are [Eve83, FPH00, FPH02, MD02, Zho04, WBZ04, FPH04]. These protocols require three messages in their exchange sub-protocols, compared to optimistic protocols for generatable items that require four messages. It has been shown in [Sch00] that three messages is the minimum number of messages in exchange sub-protocols, given that the TTP is allowed to participate in the dispute resolution phase, while this number is four if the TTP is not allowed to do so. Of course, requiring the TTP's intervention in the evidence verification phase is a drawback for these protocols: Evidences carry no meaning until the TTP declares that they are not revoked. [14]

Most FE protocols assume that the items subject to exchange are *idempotent* [Aso98], meaning that receiving (or possessing) an item once is the same as receiving it multiple times. For example, once Alice gets access to Bob's signature on a contract, receiving it later does not add anything to Alice's knowledge. The idempotency assumption reflects the mass reproducibility of digital items. However, there exist protocols for exchanging

[14] Such protocols are sometimes called *non-monotonic* [Ate04].

digital non-idempotent items. Electronic vouchers [FKT+99, FE03] are prominent examples of non-idempotent items. Depending on the implementation, right tokens in digital rights management systems can as well be considered as digital non-idempotent items, e.g. see [CIK+06, TKJ07]. The current approach to securely use non-idempotent items is to limit their distribution to trusted computing devices, which are nowadays becoming more prevalent. Protocols for handling non-idempotent items, being FE protocols or not, usually require that items are neither created nor destroyed in the course of the protocol, e.g. see [FKT+99, TIHF04]. This resembles the *money atomicity* property in electronic commerce, stating that money is neither destroyed nor generated in exchanges [Tyg96].

Using trusted devices in FE is not limited to exchanging non-idempotent items. These are used for exchanging idempotent items as well, mainly in order to increase protocols' efficiency or flexibility. Examples are [TMI+06] to reduce the number of messages to three in the optimistic sub-protocol, [VPG01] for exchanging time-sensitive items, and [TMH06] for optimistic exchange of non-revocable, non-generatable items (recall that optimistic FE requires that at least one of the items be either revocable or generatable [SW02]). [15] See also [AV04, AGGV05, FFD+06, GR06] on using trusted devices in FE.

Several results regarding optimal efficiency of asynchronous two-party optimistic CS and CEM protocols have been derived in [PSW98, Sch00]. The main results regarding the optimal number of messages in exchange sub-protocols are mentioned above: Three messages when the TTP is allowed to intervene in the dispute resolution phase, and four messages otherwise. Therefore, protocols which require only three messages in the exchange sub-protocol and do not rely on TTP's intervention in the dispute resolution phase are not fair. For instance, the protocols of [Mic03] do not provide timeliness.

The authors of [PSW98] also show that the TTP needs to be stateful (i.e. to keep states of disputed exchanges) to guarantee fairness in asynchronous optimistic protocols. From a practical point of view, this result is of great relevance: Optimistic FE not only requires TTPs for recovering from unfair transient states, it needs TTPs which maintain persistent databases, containing the states of disputed exchanges, for virtually an indefinite amount of time. Naturally, in long runs, TTPs may crash or be compromised. [16] Mechanisms to limit the damages of these defects are described below. Before that, we remark that the optimistic protocols with stateless TTPs are either unfair, such as [Mic03, Ate04, NZB04] which do not provide timeliness [17], or rely on synchronous communication channels, such as [ES05b].

To demotivate malicious TTPs from cheating on protocol participants, Asokan introduces

[15] The protocol of [VPG01] does not provide timeliness, as is pointed out in [Vog03], and the protocol of [TMH06] is susceptible to a replay attack (we skip describing the attack, as it would require a detailed description of the protocol, and the attack is also rather obvious). The ideas behind these protocols can however be salvaged with some changes.

[16] The notion of compromisable trustee may seem to be paradoxical. We note that being *trusted* does not imply being *trustworthy*, e.g. see [Gol06b].

[17] These protocols in fact require channels which can buffer messages for virtually an indefinite amount of time, thus merely delegating the "stateful-ness" to a different entity.

protocols in which TTPs are *verifiable* [Aso98]. Given that corrupted TTPs do not simply disappear, in a protocol with verifiable TTP, wronged participants can prove TTP's misbehaviour to an external court. Verifiability and transparency of TTPs are however not mutually attainable as is noted by Asokan, e.g. see [GJM99] for a concrete protocol where these two requirements clash.

To reduce the dependency of protocols on availability and sanity of one TTP, distributed TTPs can be used. In [AdG01] parts of the TTP's job are delegated to intermediary semi-trusted agents to reduce the TTP's burden, and in [SXL05, RRN05] secret sharing schemes are used so that, to subvert the protocol, an attacker needs to compromise several TTPs. Note that distributed TTPs in general need to run some atomic commit protocol to ensure the consistency of their (distributed) database. We remark that atomic commit protocols are nearly as expensive as FE, cf. [Tyg96, Tan96b, LNJ01, AFG$^+$04]. There are several alternatives to FE which do not need TTPs at all, but can only provide a weak notion of fairness. Below we discuss two of them.

The concept of *rational exchange* of Syverson [Syv98] seeks to achieve fairness, with no TTPs, assuming that the parties are rational, i.e. they try to maximise their benefits. This assumption is in contrast to the pessimistic view prevalent in the security community that honest parties should be protected even from self-damaging attackers. The idea is "not to enforce compliance with the protocol, but to remove incentives to cheat", cf. [Jak95]. A few scenarios in which rational exchange can be of practical use are mentioned in [Syv98].

Game theory can provide valuable insights into the properties of exchange protocols, when assuming that their participants are rational agents, rather than categorising them as malicious and honest parties, who blindly act regardless of their interests. For more on this approach see [San97, BH99, SW02, IIK02, CMSS03, BHC04, IZS05, ADGH06, TW07a].

Concurrent signatures proposed in [CKP04], and further investigated in [SMZ04, WBZ06, TSS06], provide a weak alternative to fair exchange. These generally do not require any TTP interventions. The idea is that Alice and Bob produce two ambiguous signatures which become bound to their corresponding signers only when a *keystone* is released by Alice. The main shortcoming of the construct is that Bob has no control over the termination of the protocol, and, moreover, Alice can secretly show Bob's signature to other parties before publishing the keystone. A few scenarios in which this level of fairness is adequate are mentioned in [CKP04].

To conclude this section, we point out some of the resources which can be of use when designing FE protocols. Many of the prudent advices [AN96] and attack scenarios known for authentication and key distribution protocols [Car94, CJ97] are pertinent to FE protocols as well. Papers specifically focusing on FE are unfortunately scarce.

We note that compilations of FE protocols are almost non-existent, [KMZ02] being a notable exception. New protocols are constantly devised with subtle differences between their assumptions, methods and goals, thus making it difficult to oversee general techniques. As of design methodologies, [Aso98, PVG03] discuss constructing generic FE protocols

and [GRV05] provides templates for conservative NR protocols. The collections of attacks on NR and CEM protocols, presented respectively in [Lou00] and [SWZ06], give designers an opportunity to assess their new protocols against known attacks. These are however not well classified, and in particular flaws stemming in the interaction between protocols and cryptographic apparatus used in them are mostly omitted (see [DR03] for an example of such attacks on FE protocols). Concerning attacks on multi-party CS, [MR06] introduces a startling attack that demonstrates the subtlety of these protocols (see also [CKS04, MKR05]). Optimality results, although known for two party FE protocols [Sch00], in multi-party cases are yet to be investigated. We are not aware of any comprehensive survey on existing formal techniques for verifying FE protocols (see [Mea03] on formal verification of security protocols in general). This would be highly desirable for practitioners.

Chapter 3

A certified email protocol using key chains

In this chapter, as an example of FE protocols, a certified email protocol is designed and thoroughly studied.

3.1 Introduction

Alice wants to send an email to Bob. She wishes to receive an evidence of receipt when Bob receives (and is able to read) the email. Bob is willing to send back an evidence of receipt to Alice only if he receives an evidence of origin along with Alice's email. Certified email (CEM) protocols are to provide such services.

There exist several CEM protocols in the literature (see § 2.2.2 for a survey). Among various value-added CEM protocols we mention [Mic97] on CEM with transparent TTPs, [KM01] on CEM with no selective receipt, [Ate04] on passive recipient CEM and [SZW05] on CEM with temporal authentication. Multi-party CEM has also attracted considerable attention [ASW96, KM00, FPH02, KH06]. See also [SWZ06] for a recent review on common pitfalls in designing CEM protocols. Below, we motivate our CEM protocol.

In order to achieve strong fairness, asynchronous optimistic CEM requires stateful TTPs, see § 2.2.2. Therefore, the amount of information that has to be stored by the TTP, virtually for an indefinite amount of time, is a serious concern in these protocols. In this chapter we introduce an asynchronous optimistic CEM protocol, with stateless recipients, that aims at reducing the TTP's storage requirements using *key chains* [Lam81], while guaranteeing strong fairness. Intuitively, a key chain is a sequence of keys such that each key is derived by applying a function to the previous one.

The proposed protocol is based on a practical aspect of CEM exchanges: Once two participants have started exchanging emails, usually several emails are exchanged between them. This observation motivates our proposed CEM protocol which imposes an initialisation overhead, but is more efficient for exchanging several emails. Except for the use of key chains, our proposed protocol is conceptually similar to the existing FE protocols such as [ZG97a, ASW98a, CCT05].

Road map In § 3.2, we present the assumptions on which our protocol is based, along with some notations. The design goals are described in § 3.3. The protocol is presented in two steps: First we give a high level description of a naïve design of the protocol in § 3.4. By

first describing this, we show the main idea underlying the protocol. The naïve design is not efficient and in § 3.5 we explain an efficient version of the protocol in detail. In § 3.6 we discuss the security of the proposed protocol and analyse how it achieves the design goals. § 3.7 concerns practical issues and implementation considerations of the protocol. § 3.8 concludes the chapter with comparing our protocol with existing schemes.

3.2 Notations and assumptions

In the following, $\mathcal{M}$ denotes the content of the email being exchanged.

Hostile environment We assume the $\mathcal{DY}$ attacker model, see § 2.1. The attacker controls the communication network and can also compromise any protocol participant, except for trusted entities. The protocol has to protect the interests of the honest participants (those who faithfully follow the protocol) in this hostile environment.

Communication channels assumptions We assume resilient communication channels between each participant and the TTP. A message inserted into a resilient channel is eventually delivered (cf. § 2.2.2). The channels connecting non-trusted protocol participants are however under complete control of attackers. Therefore, no assumption is made on delivering messages over these channels; in particular messages can get lost.

Cryptographic notations and assumptions We assume ideal cryptographic apparatus à la Dolev and Yao [DY83] as is described below (see also § 2.1.1). A message m encrypted with the symmetric key k is denoted $\{m\}^s_k$, from which m can be extracted only using k. Moreover, it is assumed that k cannot be derived from $\{m\}^s_k$ alone. We assume the existence of a public key infrastructure. The notations $pk(p)$ and $sk(p)$ represent the public and private keys of entity p, respectively. It is assumed that everyone has access to $pk(p)$ for any entity p, and $sk(p)$ is initially only possessed by p. In asymmetric encryption we have $\{\{m\}^a_{sk(p)}\}^a_{pk(p)} = \{\{m\}^a_{pk(p)}\}^a_{sk(p)} = m$. Encrypting with a private key denotes signing. We also assume access to an ideal secure hash function h (e.g. see [MVO96]) that satisfies:

- One-wayness: Given a $h(m)$ for which m is not known, it is infeasible to compute m' such that $h(m') = h(m)$.
- Collision-resistance: It is infeasible to find m and m', such that $h(m) = h(m')$, while $m \neq m'$.

For $\{h(m)\}^a_{sk(p)}$, we write $(m)_p$.

TTP assumptions We assume T is a trusted entity. T maintains a secure database, with entries of the form $\langle X, Y, Z, W\rangle$, where X and Y are participant identities, Z is a key and W contains a random number. Associated with each such entry, T stores a linked list, initially of length zero. Each element of such a linked list is of the form $(i, status(i))$, where $i \in \mathbb{N}$ and $status(i)$ stores the status of resolved exchanges. A special flag å indicates an aborted

exchange, and for recovered exchanges, a hash value (described in § 3.5.3) is stored. Thus, $status(i) \in \{0,1\}^\ell \cup \{\text{å}\}$, for a finite ℓ. We let å $\notin \{0,1\}^\ell$.

Idempotency assumption Exchanged items, i.e. emails and evidences, are assumed idempotent. Therefore, to receive an email or receipt twice is not different from receiving it once (meaning that it is not considered as an attack).

3.3 Design goals

One of the most fundamental requirements for CEM protocols is *non-repudiation*. Non-repudiation guarantees that an agent cannot deny having sent or received an email message, if it has actually done so in the course of the protocol. To achieve this, protocol participants usually collect evidences, evidence of origin (EOO) and evidence of receipt (EOR), which can later be presented to a judge.

The second requirement for CEM protocols is to satisfy the *fair exchange* properties. Fair exchange consists of three properties:

- *Timeliness* states that an honest participant can unilaterally, or with the help of the TTP, terminate the protocol run, i.e. reach a state where it has no further pending operations to perform in that protocol run.
- *Effectiveness* states that if honest A and B engage in the protocol and are willing to exchange emails for receipts, then, assuming that communication channels do not excessively delay delivering messages, the protocol will terminate in a state where B has received the email content $\mathcal{M}$ and EOO, and A has received EOR.
- *Fairness* states that if A terminates the protocol in a state where she possesses EOR, then B terminates (or has terminated) the protocol in a state where he possesses both $\mathcal{M}$ and EOO; and if A terminates the protocol in a state where she does not possess EOR, then B terminates the protocol in a state where he possesses neither $\mathcal{M}$ nor EOO.

Confidentiality is another requirement for CEM protocols which states that the exchanged email content should not be revealed to anyone (including the TTP), except to the intended receiver. Our proposed protocol does not directly address confidentiality. However, if confidentiality is desired, the exchanged email content $\mathcal{M}$ can be substituted with the actual email content encrypted for the receiver using his public key or a shared secret key.

3.4 A naïve protocol

In most existing NR and CEM protocols, the initiator uses a separate key to encrypt the email to be sent, for each single exchange. If the exchange goes amiss, the parties can resort to a TTP, that will store the key along with some other information about the exchange, such as

involved parties, a hash value of the email content and an exchange label (e.g. see [GRV05]). This information is stored virtually for an indefinite amount of time, see § 3.8.

We aim at reducing the amount of information stored by the TTP using key chains. A chain of keys is a sequence of keys $K_0, \ldots, K_n$ such that $K_i := H(K_{i-1})$, for $i > 0$, where H is a publicly known one-way collision-resistant hash function. The key K_0 is the chain's *seed*. The key chain is initiated by the initiator Alice who chooses a seed K_0 and shares it with the TTP. They moreover agree on the maximum number n of exchanges that Alice can perform using the chain. Afterwards, Alice traverses the chain backwards and she uses K_{n-i} (for $i \leq n$) to encrypt the message in the i^{th} exchange. Since the function H is assumed to be one-way, i.e. given $H(x)$ it is infeasible to compute x, the key used in i^{th} exchange remains unknown to the receiver Bob unless he knows one of the K_j for $j \leq i$. However, since Alice traverses the chain backwards, the keys seem to be fresh and independent.

The i^{th} exchange starts with Alice sending an email to Bob, encrypted using K_{n-i}, a key that Bob does not know (yet). Bob commits to the exchange by acknowledging the reception of the encrypted email. Afterwards Alice sends Bob the key and, finally, Bob acknowledges the reception of the key. In this scenario the TTP interferes only when a party does not receive the message he or she expects. Intuitively, when a party can prove that the opponent has committed to the exchange, then the TTP provides that party with the encryption key along with some affidavits. When key chains are used, in order to produce the key used in any resolved exchange, the TTP only needs to store the seed. The TTP's storage requirements are thus reduced.

Obviously, this protocol is not purely optimistic because Alice needs to set up a chain with the TTP. However, if the number n of exchanges is large enough, then the gained reduction in required storage space of the TTP will, in many practical applications, compensate the overhead of the initial setup phase. But, one single problem undermines the efficiency of this protocol: It is costly to abort an exchange. This is because of the following situation: Assume that exchange number i is aborted. This means that K_{n-i} is not revealed to Bob, but he gets hold of the encrypted email. This can happen for instance when Alice sends the encrypted email to Bob, but afterwards aborts the exchange because Bob is slow in replying. Now if the protocol proceeds and the $(i+1)^{th}$ session terminates successfully, then Bob learns $K_{n-(i+1)}$. Because of the way the chain is constructed, Bob can easily compute K_{n-i} (that is $H(K_{n-(i+1)})$) and decrypt the email content of the aborted session at will. Fairness is thus violated in this situation. Therefore, if an exchange is aborted, Alice needs to abandon using the rest of the chain altogether and set up a new key chain with the TTP. This can potentially impose a huge efficiency penalty on the protocol. The next section describes a way to circumvent this problem.

3.5 The main protocol

Here we describe our asynchronous optimistic fair CEM protocol. This protocol uses keys in a key chain for encrypting emails that are exchanged. Once this chain has been initialised, emails can be encrypted and exchanged, each time with a new key of the chain. The protocol also provides a way for the initiator to revoke an entire key chain.

Each exchange (or attempt to exchange) that uses the optimistic protocol (and possibly also the recovery and abort protocols) is called a *protocol round*. An initialisation phase followed by a number of protocol rounds is called a *protocol session*. After the initialisation phase, the initiator can send emails to the responder. Each protocol session belongs to one unique initiator-responder pair. However, the protocol naturally allows concurrent sessions. An agent can thus be involved in different sessions with different partners at the same time.

3.5.1 Initialisation

The initiator A chooses a random key K_0, the seed of the key chain, from a large key domain $\mathcal{K}$. Let $H : \mathcal{K} \to \mathcal{K}$ be a publicly known one-way collision-resistant hash function and $G : \mathcal{K} \to \mathcal{K}$ be a publicly known acyclic function, i.e. $\forall k, i.\ G^i(k) \neq k$. [1] For $i \geq 0$, we define $K_{i+1} := G(K_i)$ and $K'_i := H(K_i)$. We require that H and G do not commute, i.e. given a $H(k)$ for which k is unknown, it is infeasible to compute $H(G(k))$ (see § 3.7).

$$\begin{array}{ccccccccc} K_0 & \overset{G}{\longmapsto} & K_1 & \overset{G}{\longmapsto} & \cdots & \overset{G}{\longmapsto} & K_i & \longmapsto & \cdots \\ \downarrow H & & \downarrow H & & & & \downarrow H & & \\ K'_0 & & K'_1 & & & & K'_i & & \end{array}$$

Figure 3.1: *Double key chain*

The sequence $K'_0, K'_1, \ldots$ of keys is used for encrypting the emails which are to be exchanged. Clearly any K'_i can be calculated from K_0 using H and G (see figure 3.1). To initialise a session, A sends the seed K_0 and the identity of the potential responder (of the session) B to the TTP T. [2]

$$\begin{array}{lll} 1^0. & A \to T : & \{A, B, K_0, nc\}^a_{pk(T)} \\ 2^0. & T \to A : & sid, \mathsf{cert}, (nc, \mathsf{cert})_T \end{array} \tag{3.1}$$

where the certificate $\mathsf{cert} := (A, B, sid)_T$ and nc is an unpredictable *nonce* chosen by A to prevent replay attacks (see § 3.7), and sid is a unique session identifier chosen by T.

[1] For G to be acyclic, we require an infinite $\mathcal{K}$. In practice, usually infinite key spaces are approximated with a sufficiently large key space, cf. [MVO96].

[2] We use superscripts for message numbers in order to unambiguously refer to them in the text.

A nonce is a *number used once*. A nonce is thus always fresh. An extra requirement for nonces is unpredictability, which can be achieved via, e.g., randomness, cf. [AN96].

T stores entries of the form $\langle initiator, responder, seed, sid\rangle$, where $seed$ is the seed chosen by A. The key chain rooted at $seed$ can be used for sending CEMs from $initiator$ to $responder$. When T receives message 1^0, it looks for the entry $\langle A, B, K_0, *\rangle$ in its database. If $\langle A, B, K_0, *\rangle$ is not already present in its database, then T chooses a fresh session identifier sid and adds $\langle A, B, K_0, sid\rangle$ to the database. The TTP then sends back message 2^0, a confirmation of that it approves this session. If $\langle A, B, K_0, *\rangle$ already exists in the database, T ignores the request (and sends back an error message to A).

Differently than in the protocol sketch in § 3.4, in the main protocol it is not needed to specify the maximum number n of exchanges. In the protocol sketch, since Alice needed to traverse the chain backwards, n was used to specify the start point of the chain traversal. In the main protocol the entire chain is obscured using H. So it is not needed to put any order on the key chain traversal. Hence, it is not needed to specify n.

3.5.2 Exchange sub-protocol

Each protocol round has an order number i, which initially is 0 and can arbitrarily grow. After each round the initiator A increments i. The i^{th} protocol round is as follows

$$\begin{array}{lll} 1^{ex}. & A \to B: & A, B, T, i, sid, h(K'_i), \{\mathcal{M}\}^s_{K'_i}, EOO_M, \mathsf{cert} \\ 2^{ex}. & B \to A: & EOR_M \\ 3^{ex}. & A \to B: & K'_i \\ 4^{ex}. & B \to A: & EOR_{K'} \end{array} \tag{3.2}$$

Where

- $EOO_M := (B, T, h(K'_i), \{\mathcal{M}\}^s_{K'_i}, i, sid)_A$
- $EOR_M := (EOO_M)_B$
- $EOR_{K'} := (A, K'_i, \{\mathcal{M}\}^s_{K'_i})_B$

Here h is a secure hash function, which can be chosen to be H. In the first message, A sends the encrypted email content $\{\mathcal{M}\}^s_{K'_i}$, the hash value of the encryption key $h(K'_i)$ and the session certificate $(A, B, sid)_T$. The responder B checks the correctness of the message and commits himself to receive the email by sending message 2^{ex}, if he trusts T. Then A sends the key K'_i. Agent B checks that this key matches the hash value of the key that he received in message 1^{ex}. Finally, if the key is correct, B sends a confirmation of having received the key. The number i is only used implicitly by B when he resolves the protocol round.

3.5.3 Recovery sub-protocol

The initiator A may run the recovery protocol after having received message 2^{ex} in protocol 3.2, by presenting EOR_M to the TTP. This shows that A has actually sent EOO_M to B, ensuring that B is also able to receive a recovery token for that exchange. Agent A typically runs the recovery protocol to complete the EOR (as defined in § 3.5.5 below), if she does not receive message 4^{ex}. The responder B may run the recovery protocol after receiving message 1^{ex}, in order to get the encryption key. The recovery protocol initiated by $P \in \{A, B\}$ starts with the following message:

$$1^r. \quad P \rightarrow T: \ f_r, A, B, h(K'_i), h(\{\mathcal{M}\}^s_{K'_i}), i, sid, EOR_M \tag{3.3}$$

where f_r is a flag used to identify the recovery request. On receiving this message, T performs the following tests:

- T checks if the signatures in the message are genuine and if its own identity is given as the designated TTP.
- T checks whether there is an entry in its database matching $\langle A, B, *, sid\rangle$.
- If the previous tests succeed, then the result of the query would be a unique entry $\langle A, B, K_0, sid\rangle$ (see § 3.5.1). Subsequently T uses the retrieved K_0 to check whether $h(H(G^i(K_0)))$ matches $h(K'_i)$ in the message.

If the results of all these tests are affirmative, then T checks whether round i has already been resolved or not. For each key chain (corresponding to one single entry $\langle A, B, K_0, sid\rangle$ in T's database) and each exchange $i \geq 0$ that is resolved at T, T stores whether that exchange has been recovered or aborted in a status variable $status(i)$ (cf. § 3.2). If $status(i)$ has not been initialised in T's database, it sets $status(i) := h(\{\mathcal{M}\}^s_{K'_i})$. [3] Then T proceeds as if $status(i)$ had already been set for the exchange, as is described below.

If $status(i)$ has already been initialised in the database, T sets $v := \perp$ if $status(i) =$ å, and sets $v := K'_i$ in case $status(i) = h(\{\mathcal{M}\}^s_{K'_i})$. Then T sends the following message and terminates this resolve transaction.

$$2^r. \quad T \rightarrow P: \ v, (A, B, h(\{\mathcal{M}\}^s_{K'_i}), v, i, sid)_T \tag{3.4}$$

The message $(A, B, h(\{\mathcal{M}\}^s_{K'_i}), \perp, i, sid)_T$, where $\perp$ is a special flag that denotes an aborted exchange, serves as an abort token. When P receives this message, it can safely quit the protocol round. The message $K'_i, (A, B, h(\{\mathcal{M}\}^s_{K'_i}), K'_i, i, sid)_T$ serves as a recovery token for P (see evidences in § 3.5.5).

[3] Note that if the hash function h produces hash values of ℓ bits length, i.e. $h : \{0,1\}^* \rightarrow \{0,1\}^\ell$, then we have $status(i) \in \{\text{å}\} \cup \{0,1\}^\ell$.

If $\neg(status(i) = h(\{\mathcal{M}\}^s_{K'_i}))$ or any of the tests mentioned above fails, then T ignores the recovery request and sends back the following message, tagged with the special flag error:

$$2^r. \quad T \rightarrow P: \ \mathsf{error}, (\mathsf{error}, m^r)_T \tag{3.5}$$

where m^r is the content of the message sent in step 1^r (protocol 3.3). This indicates a misbehaviour, and based on this message, P can quit the protocol round. Note that P needs to be sure of the freshness and authenticity of this message, which is attained by T's signature on m^r. This is needed to prevent simple replay attacks.

3.5.4 Abort sub-protocol

The initiator A may abort an exchange at any stage, provided that the exchange has not been recovered already. Typically A aborts if she does not receive message 2^{ex}. On the other hand, the responder B can never explicitly request the TTP to abort an exchange that has been initiated by A. To abort an exchange, A sends T the following message:

$$1^a. \quad A \rightarrow T: \ f_a, A, B, h(\{\mathcal{M}\}^s_{K'_i}), i, sid, \mathsf{abrt} \tag{3.6}$$

where $\mathsf{abrt} := (f_a, B, T, \{\mathcal{M}\}^s_{K'_i}, i, sid)_A$ and f_a is a flag identifying the abort request.

On receiving this message, T checks A's signature on abrt and its own identity in the message, and it queries its database with $\langle A, B, *, sid\rangle$ only if they match. The result is a unique $\langle A, B, K_0, sid\rangle$ (see Initialisation phase). Then T checks whether session i has already been resolved, by checking whether $status(i)$ has been initialised in its database or not. If not, T sends back the following message:

$$2^a. \quad T \rightarrow A: \ (A, B, h(\{\mathcal{M}\}^s_{K'_i}), \perp, i, sid)_T \tag{3.7}$$

This message serves as an abort token and when A receives it, A can safely quit the protocol. Then, T sets $status(i) :=$ å and terminates this resolve transaction. Similarly, if $status(i)$ has already been set in T's database as $status(i) =$ å, then T sends the above message and terminates the resolve transaction.

If $status(i)$ has already been set in T's database indicating a recovery, i.e. $\neg(status(i) =$ å), then T tests whether $status(i) = h(\{\mathcal{M}\}^s_{K'_i})$. If the test succeeds, T sends the following message and terminates this resolve transaction:

$$2^a. \quad T \rightarrow A: \ (A, B, h(\{\mathcal{M}\}^s_{K'_i}), K'_i, i, sid)_T \tag{3.8}$$

This message constitutes a recovery token for A, for this exchange.

If $\neg(status(i) = h(\{\mathcal{M}\}^s_{K'_i}))$ or if $\langle A, B, *, sid\rangle$ does not exist in T's database, or if any of the tests mentioned above fails, then T ignores the abort request and sends back an error message:

$$2^a. \quad T \rightarrow A: \ \mathsf{error}, (\mathsf{error}, \mathsf{abrt})_T \tag{3.9}$$

Note that an abort token does not necessarily mean that an exchange has not finished successfully, since A can abort an already completed exchange. An abort token merely indicates that T will never issue a recovery token (hence releasing the key) for that particular protocol round, uniquely identified with sid and i.

3.5.5 Evidences and dispute resolution

In case of a dispute, the parties can present evidences to an external judge. We note that each protocol round (of each protocol session) has an associated EOO and EOR.

The evidence of receipt EOR, desired by A, consists of

$$A, B, T, M, i, sid, K'_i, \mathsf{cert}, EOR_M, EOR_{K'},$$

if it is obtained by running the exchange protocol. If A uses the recovery or abort protocols, then the last two elements $EOR_M, EOR_{K'}$ are replaced by the recovery token from the TTP, i.e. $(A, B, h(\{\mathcal{M}\}^s_{K'_i}), K'_i, i, sid)_T$.

The evidence of origin EOO, desired by B, consists of

$$A, B, T, M, i, sid, K'_i, EOO_M.$$

A judge settles a dispute by simply checking whether the EOR or EOO presented by the disputing parties are genuine. We emphasise that abort tokens have no weight in these evidences. Therefore, having an abort token does not override or revoke having a recovery token. The purpose of the abort protocol is solely to guarantee timeliness for the initiator.

3.5.6 Revoking compromised key chains

In practice it may happen that A's computer is compromised and the key chain seed is revealed to an attacker. In such situations, A might want to revoke the key chain she has set up with the TTP. [4] Therefore, the protocol allows A to ask the TTP, at any moment, to mark her key chain as *obsolete*:

$$A \rightarrow T: \ f_o, \mathsf{cert}, (f_o, \mathsf{cert})_A \tag{3.10}$$

Here f_o is a flag that denotes a request to mark the chain identified by cert as obsolete. Upon receiving this message, T checks A's signature and if cert is a genuine certificate from T to A, and, only if this is the case, marks the entry $\langle A, B, *, sid\rangle$, which is unique, as obsolete. Marking an entry as obsolete means that T will not recover or abort protocol rounds connected to that entry any more. But T will behave as usual if it is queried about a protocol round for which a $status$ value has already been set. This mechanism ensures that A cannot

[4] We do not discuss methods for revoking A's private key, in case it is compromised by the attacker. Revoking keys in PKIs has extensively been studied, e.g. see [BDTW01].

cheat B by first resolving an exchange and then marking the chain as obsolete (purporting that Bob would not be able to recover exchanges that use an obsolete chain).

We remark that revoking a key chain does not protect contents of the emails that belong to previously aborted protocol rounds: If the attacker records the communications between A and B, and subsequently compromises the key chain that A has used to send CEMs to B, then it can read the contents of emails from all previous protocol rounds (including those) which were aborted.

A prudent Alice would only store the last elements of the key chains that she uses. For instance, after encrypting an email using K'_i, Alice would only need K_{i+1} for future encryption, thus $K_0, \cdots, K_i$ can be permanently discarded. Then, if G is a one-way function, the attacker does not learn the value of the keys used in previously aborted protocol rounds [5] by breaking into Alice's computer. This defensive strategy is not effective if G is reversible.

3.6 Security analysis

In this section we justify the protocol by informally showing that it achieves the design goals described in § 3.3. A difficulty in formal (finite state) verification of the protocol is handling the key chains: Even in a finite session model, a malicious Alice can select to use keys arbitrary deep in the chain, effectively making the state space infinite. Various abstraction and data independence methods can in principle be used to alleviate this difficulty, e.g. see [Laz99, HS06]. These however need to be carefully adapted to the setting of our protocol. Formal verification of this protocol is hence left as future work.

Before focusing on the goals of the protocol, we state two relevant properties of the TTP's database.

1. Persistence: If, for a certain sid, T assigns å or some $h(\mathcal{M})$ to $status(i)$, then this value is never changed in the future. Persistence follows from the TTP's specification according to the protocol: T assigns values to $status(i)$, only if $status(i)$ does not already have a value. The logic of the TTP is summarised in figure 3.2.

   For each i (of each sid), the finite state (Mealy) machine of the TTP may be at one of the following states: Unresolved s_U, aborted s_A and recovered s_R. If the TTP receives an abort request while being at state s_U, it grants the request (i.e. sends back an abort token) and moves to state s_A, and similarly for other cases. In figure 3.2, a and r stand for valid abort and recovery requests, and A and R stand for the corresponding abort and recovery tokens, respectively.

2. Consistency: If, for a certain sid, T assigns $status(i)$ with å, then $\forall \mathcal{M}.\ status(i) \neq h(\mathcal{M})$, and vice versa. Consistency follows from the definition of å, see § 3.2.

[5] The keys that belong to successful exchange rounds are ultimately revealed by Alice (or the TTP), and thus the attacker does not need to break into Alice's computer to obtain them.

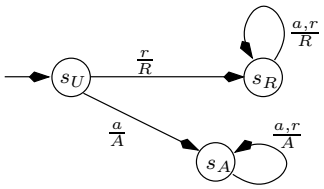


Figure 3.2: *The Mealy machine of the TTP*

In the following, we discuss why the protocol's design goals are achieved. For convenience, below, A and B denote the initiator and the responder parties, respectively.

- Non-repudiation: If A possesses EOR (see § 3.5.5), then she either has $EOR_{K'}$ or the recovery token for that round. In the first case, she can show that B has received both $\{\mathcal{M}\}^s_{K'}$ and K'. In the second case, B has $\{\mathcal{M}\}^s_{K'}$ and either he has or he can receive K' by using the recovery protocol. Agent B is therefore able to extract $\mathcal{M}$. Moreover, B cannot deny that he is able to obtain $\mathcal{M}$ because of his signatures in EOR. Similarly, if B possesses EOO, he can show that A has indeed sent K' and $\{\mathcal{M}\}^s_{K'}$, because of here signatures in EOO_M. Note that it is only A (or the TTP on behalf of A) who is able to generate K'.

- Timeliness: The agents A and B terminate a protocol round either by completing the exchange protocol, or by executing the recovery or abort protocol. Agent A can run the abort protocol at any time. She can also run the recovery protocol after she has received the second message in the exchange protocol. Agent B can recover the protocol after he has received the first message in the exchange protocol. Termination is guaranteed by the fact that the channels to the TTP are resilient. Note that termination of B depends on that he has the identity of the designated TTP signed by A in the first message of the exchange protocol (as it is in protocol 3.2). This is because A would otherwise be able to cheat B by resolving at a TTP which is not known (or trusted) to B, hence leaving B in an unfair situation.

  To show that the degree of fairness does not decrease for an honest participant after termination, it is enough to show that, if A (B) has terminated and not received EOR ($\mathcal{M}$ and EOO), then B (A) will not receive $\mathcal{M}$ or EOO (EOR). But, if A (B) has terminated and not received EOR ($\mathcal{M}$ and EOO), then we infer that the protocol round was aborted, and in an aborted round, B (A) cannot learn $\mathcal{M}$ or EOO (EOR).

- Effectiveness: If A and B faithfully follow the exchange protocol and messages are not excessively delayed, A receives EOR iff B receives $\mathcal{M}$ and EOO.

- Fairness: As mentioned above (effectiveness), if the exchange protocol terminates normally then A receives EOR and B receives $\mathcal{M}$ and EOO. Thus, in this case the pro-

tocol round terminates in a fair state. If the exchange protocol is (accidentally or maliciously) interrupted, then the agents resolve the round using the recovery and abort sub-protocols. There are three main cases to consider. For each one of them we will show that both agents can resolve the round, and when they have done so a fair state is reached:

1. Both agents run the recovery protocol. Agent A can only recover the protocol round by presenting EOR_M to the TTP, hence proving that B has indeed received EOO_M in the interrupted exchange protocol. In this case, because of persistence and consistency of the TTP's database, B can also recover and receive the encryption key used in that round.
2. Agent A aborts the round and afterwards B runs the recovery protocol. Agent A can abort a round at any stage, unless that round has already been resolved. Afterwards when B runs the recovery protocol, the TTP responds with a message containing a special flag that denotes that the round was already aborted (because TTP's database is persistent and consistence). Thus, none of the agents get their evidences, and B does not receive the encryption key.
3. Agent B recovers the round and afterwards A runs the abort protocol. When A runs the abort protocol (and the round is already recovered), the TTP responds with the recovery token (because TTP's database is persistent and consistence). Now both agents have their evidences, and B has received the encryption key and is able to obtain $\mathcal{M}$.

The agents can run the recovery and abort protocols also after an normal termination of the exchange protocol. But this has no effect on the evidences EOR and EOO.

It may seem as if it could be a problem that A can reuse a key K'_i that has been used in a previous (possibly recovered or aborted) protocol round, to initiate a new round. But, as we will show here, it is in A's own interest to never reuse keys. Since $EOR_{K'}$ and the recovery token contain $h(\{\mathcal{M}\}^s_{K'})$, A needs either to send K' in the exchange protocol or to recover the exchange in order to receive $EOR_{K'}$ or the recovery token and complete the EOR. In the first case, the exchange protocol terminates successfully, leaving A and B in the same fair state. In the second case, there are three possibilities: (1) The key K'_i has been used in a protocol round that terminated normally: In this case, the protocol executes normally, i.e. as if i is a fresh (not reused) index. (2) The key K'_i has been used in a protocol round that was recovered: If B is honest, B will also recover, and when they recover, both A and B receive an error message (unless the email content is actually exactly the same as what has already been recovered). This leaves A and B in the same (fair) state. (3) The key K'_i has been used in a protocol round that was aborted: A receives neither the recovery token nor the last message from B. Thus, A cannot collect an EOR, neither can B collect EOO.

Potentially A could abort a completed exchange and then start a new protocol round with the same email content and key. In the new round, if A fails to continue the exchange protocol after receiving EOR_M, B receives an abort token when he tries to resolve, since the TTP has actually stored that round as aborted. But, because of the idempotency assumption (see § 3.2), A does not gain anything more than what she had before reusing the key and the email content. Moreover, the abort token does not override the EOO that B has already collected. Thus the level of fairness achieved by B is not decreased.

We finally remark that if A maliciously uses a key that has been revealed to B earlier (or later), then B can easily (by violating the protocol) obtain an EOO without sending message 2^{ex}.

3.7 Implementation considerations

Security of the initialisation phase We motivate that in practice it is reasonable to require A to sign parts of message 1^0 of the initialisation phase (§ 3.5.1) as

$$\begin{array}{lll} 1^{0\prime}. & A \to T: & \{A, B, K_0, nc, (K_0, nc)_A\}^a_{pk(T)} \\ 2^{0\prime}. & T \to A: & sid, \mathsf{cert}, (nc, \mathsf{cert})_T \end{array} \tag{3.11}$$

If A cannot generate unpredictable nonces with high entropy (e.g. see [CA-01]), then the initialisation protocol 3.1 can be subverted as is described below (the attacker is called Z, and $Z(X)$ means that the attacker pretends to be participant X).

$$\begin{array}{ll} A \to Z(T): & \{A, B, K_0, nc\}^a_{pk(T)} \\ Z(A) \to T: & \{A, B, L, nc\}^a_{pk(T)} \\ T \to A: & sid, \mathsf{cert}, (nc, \mathsf{cert})_T \end{array}$$

In this scenario, the attacker can send the second message only if the value of nc is predictable. The result is that T initiates a session between A and B, which A believes uses K_0 as its seed, while the seed is in fact L. If the protocol requires A to sign the nonce nc (as in protocol 3.11), then to avoid the attack, having fresh (but not necessarily unpredictable) nc would be enough, which is easier to achieve in practice.

Similarly, if A chooses K_0 poorly, an undetectable on-line guessing attack (see [DH95]) can reveal the value of K_0: The attacker would guess a key L and then ask T to initiate a session between A and B with L as its seed: $Z(A) \to T\ \{A, B, L, nc\}^a_{pk(T)}$. If $L = K_0$, then T would send an error message to Z, thus confirming Z's guess. Such an attack would be impossible if $(K_0, nc)_A$ is included in the message, as is in protocol 3.11.

Our last motivation addresses denial of service attacks. In practice, because of its resource constraints, T may not be able to accept all valid requests. If the requests are signed by their initiators, T would be able to deploy a fair quota system.

Type flaw attacks Type flaw happens when (a part of) a message is interpreted as having a different type from the intended one. In the described sub-protocol there are possibilities for type confusion which can easily be avoided by simply tagging messages and signatures with their purpose. For instance, a malicious A, exploiting B's signature on $K'_i, \{\mathcal{M}\}^s_{K'_i}$ (as $EOR_{K'}$ in protocol 3.2), could get access to B's signature on $EOO_{M'}$ in another session. This scenario can be prevented by letting $EOR_{K'} = (f_{EOR_{K'}}, A, K'_i, \{\mathcal{M}\}^s_{K'_i})_B$, and so forth. See [HLS03, LYH04] for general methods to prevent type flaws.

Constructing hash chains The non-commutativity requirement on H and G (see § 3.5.1) rules out choosing, e.g., $G(k) = k$ or $G(k) = H(k)$. This also implies that if for G a simple linear function, such as $G(k) = k+1$, is selected, then H should be non-incremental [BGG94]. Hash chains can in practice be constructed using SHA-1 hash functions [EJ01] as H. A choice for G can be MD5 [Riv92]. Other options for these would be to use various HMAC constructions [BCK96] or secure block cipher encryption algorithms.

In light of the recently discovered weaknesses of the SHA family of hash functions (e.g. see [WYY05]), we notice that collision resistance is not of paramount importance to our protocol. There are two ways to counter collision attacks. The first is to add redundancy to $\mathcal{M}$ (before encryption) in message 1^{ex}. In this way a collision between the keys k and k' can be detected because the incorrect key yields an incorrect message after decryption. The second approach is to require T, instead of the initiator A, to determine the initial key K_0.

A common problem with hash chains is that when the length of a chain is increased, in practice the chance that a collision between that chain and another one occurs increases. A standard solution (e.g. see [HJP05]) to this problem is to reduce the chance of collision by using the corresponding indexes when computing hash values. For an in-depth discussion on constructing and implementing hash chains we refer to [HJP05].

Symmetric key encryptions Symmetric encryptions of the proposed protocol can be implemented using the AES encryption standard [NIS01]. Using hash functions to construct keys enforces fixed key lengths, e.g. 160 bits if SHA-1 is used for H. We note that currently AES with 128-bit keys is considered secure. Therefore, the result of the hash function can be truncated to 128 bits to fit into the AES standard. [6]

3.8 Conclusions and related work

In this chapter, we have introduced an asynchronous optimistic CEM protocol with stateless recipients. [7] The protocol relies on key chains to reduce the storage requirements of the TTP, improving on existing schemes that achieve strong fairness. We have analysed the protocol

[6] The Rijndael algorithm, on which the AES is based, in fact allows using 160-bit keys, while the AES standard only supports 128, 192 and 256 bit key sizes [DR02].

[7] We do not require the receiver to keep any information regarding his state in the current protocol *round*, while being stateless in [Ate04] is much finer and refers to the current protocol *session*.

informally and showed that it guarantees non-repudiation, effectiveness, (strong) fairness and timeliness.

Related work We use hash chains of keys for repeated encryption in CEM protocols. The idea of using hash chains in security protocols can be traced back to [Lam81], where these are used for repeated authentication. Hash chains have later on been used in various authentication protocols, e.g. [ABC+98, PTSC00], and key management systems, e.g. [Dae98].

There is no general consensus in the literature on the requirements of CEM protocols. For example, in [ZG96a, AG02], it is not considered necessary for CEM protocols to provide EOO. In [Ate04] it is argued that timeliness for the receiver is not required in CEM protocols. Conversely, in [Mic03], timeliness for the sender is deemed unnecessary. We aim at strong fairness which guarantees timeliness for both parties, and provides EOO and EOR.

Below we study the efficiency and compare the TTP's storage requirements and the number of messages in the exchange protocol, between our proposed protocol and existing schemes. Existing schemes often require the TTP to store the key used for each single exchange along with the identities of the participants, the hash of the exchanged message and typically also a unique exchange identifier, e.g. see [GRV05] for a review. In our protocol the TTP only needs to store one seed for each chain and the status of recovered or aborted rounds. When resolving, the TTP has to perform a few hash function computations in our protocol. However, these are in general very cheap. So, when exchanging multiple certified emails, our protocol outperforms the existing asynchronous optimistic CEM protocols (that provide strong fairness) in the amount of information stored by the TTP. Note that the protocols of [Mic03, Ate04, NZB04], which only need stateless TTPs, are not strongly fair, i.e. they do not guarantee timeliness for either sender or receiver. [8] Other protocols with stateless TTPs, such as [DGLW96], are not optimistic, or rely on synchronous communication channels as in [ES05b]. In fact, it has been shown in [Sch00] that asynchronous optimistic CEM with stateless TTPs cannot provide strong fairness, see also § 2.2.2.

Concerning the number of messages in the exchange protocol, according to [Sch00], four messages is the least to achieve strong fairness for asynchronous optimistic CEM protocols. Existing CEM protocols with only three messages in the optimistic phase are not strongly fair: The protocols of [Mic03] do not guarantee timeliness of the sender. The protocols of [FPH00], [MD02] (fixing a flaw in [FPH00]), [WBZ04] (fixing a flaw in [FPH00] and [MD02]), [FPH02] and [Zho04] (fixing a flaw in [FPH02]) achieve fairness only under the rather unrealistic assumption that the cheater party collaborates with the cheated party and attends the court, in case of a dispute. In these protocols, since some of the collected evidences can conceptually be revoked based on other evidences (cf. [Eve83, PSW98]), only a weak notion of fairness is attainable. See also § 2.2.2.

[8] These protocols in fact require a channel which buffers messages for an indefinite amount of time, thus only delegating the "stateful-ness" to a different entity.

Chapter 4

Formal methods

A book which does not contain its counterbook is considered incomplete.

Jorge Luis Borges's *Tlön, Uqbar, Orbis Tertius* (1940)

Road map To formally study whether a security protocol achieves its design goals, first we need to specify the protocol in a formal language. For this purpose we use process algebra, which is the topic of § 4.1. Let us say that L specifies the protocol that is to be analysed. To verify L, we can specify the ideal behaviour of the protocol as another system L_{ideal} and use a suitable observational equivalence $\approx$ to (dis)prove $L \approx L_{ideal}$. This approach, although elegant and powerful, requires heavy human interaction, e.g. see [AG97, GR01, Can02, Hüt02].

Another approach is to specify the desired properties of the protocol in a modal logic and, then, check if L satisfies the modal formulae. We choose this approach and use μ-calculus as the property specification language, which is the topic of § 4.2. For verifying protocol behaviours against their goals we use finite state model checking, which is discussed in § 4.3.

4.1 Process algebra

For specifying protocol participants and the $\mathcal{DY}$ intruder, we use the process algebraic language μCRL [GP95], which is an extension of ACP [BK85] with abstract data types. Our results do however not depend on this choice in any crucial way, as μCRL is similar to other general purpose process algebras. What follows provides a brief introduction to μCRL, while its complete syntax and semantics are given in [GP95], see also [GR01, Fok07].

A μCRL specification consists of data type declarations and process behaviour definitions, where processes and actions can be parametrised with data. Data are typed in μCRL and types can have recursive definitions. Each non-empty data type has constructors and possibly non-constructors associated to it. The semantics of non-constructors is given by means of equations. The presence of a data sort $Bool$ of Booleans with constants T and F as constructors, and the usual connectives $\wedge$, $\vee$ and $\neg$ as non-constructors, is always assumed. We describe these concepts via an example in § 4.1.2.

The specification of a component is a guarded recursive equation that is constructed from a finite set of action labels, process algebraic operators and recursion variables. The set of

action labels is denoted Act. All members of Act, except for a designated action label τ for silent steps, may be parametrised with data to construct actions. When $a(d)$ is an action with action label $a \in Act$ and data parameter d, we write $\lambda(a(d)) = a$. For a set of actions A, we let $\lambda(A) = \{\lambda(a) \mid a \in A\}$. Operator λ is merely a means to explicitly refer to action labels.

The process algebraic operators $+$ and $\cdot$ denote nondeterministic choice and sequential composition, respectively: The process $p + q$ can behave either as process p or as process q, and the process $p \cdot q$ behaves as process p and when p terminates (if it ever does), it continues as process q. The constant δ denotes a deadlock process, i.e. a process which cannot perform any actions. Recursion variables, which can be parametrised with data, are used in the natural way, e.g. $X = a \cdot X$, with $a \in Act$, describes a process that performs action a and then recurs (performs an infinite number of a actions in sequence). A recursive equation is guarded if all its recursion variables are preceded by an action.

The summation operator $\Sigma_{d:D}\ p(d)$, where d is a free variable in process $p(d)$, provides the possibly infinite choice over a data type D. The conditional construct $p \lhd b \rhd q$, with $b : Bool$, behaves as p if $b = \mathsf{T}$ and as q if $b = \mathsf{F}$. For instance, the construct $\Sigma_{d:D}\ p(d) \lhd f(d) \rhd \delta$, with $f : D \to Bool$, forces choosing d values in $p(d)$ that satisfy f. The operator $\cdot$ has the strongest precedence, the conditional construct binds stronger than $+$, and $+$ binds stronger than Σ.

The parallel (asynchronous) composition $p \| q$ interleaves the actions of p and q. Moreover, actions from p and q may synchronise, when this is explicitly allowed by a predefined commutative associative partial function $| : Act \times Act \to Act$. Two actions can synchronise only if their data parameters are semantically equal. This implies that synchronisation can be used to represent data transfer between processes. Encapsulation $\partial_H(p)$, which renames all occurrences of actions from set H in p into the deadlock action δ, can be used to force actions into communication. For example, with $a, b, c \in Act$ and $a|b = c$, the process $(a.\delta) \| (b.\delta)$ behaves as $a.b.\delta + b.a.\delta + c.\delta$. Therefore, $\partial_{\{a,b\}}((a.\delta) \| (b.\delta)) = c.\delta$.

Next, we define labelled transitions systems, which provide a semantics for μCRL specifications.

4.1.1 Labelled transition systems

A labelled transition system (LTS) L is a tuple (S, s_0, A, Tr), where S is a countable set of states, $s_0 \in S$ is the initial state, A is a set of actions and $Tr \subseteq S \times A \times S$ is the transition relation. A transition $(s, a, s') \in Tr$, denoted $s \xrightarrow{a} s'$, intuitively indicates that the system can move from state s to s' by performing action a. L is finite if S and A are both finite.

For $a \in A$, we abuse the notation and write $a(s) = \{s' \in S \mid s \xrightarrow{a} s'\}$. The set of enabled actions at state s is defined as $en(s) = \{a \in A \mid \exists s' \in S.\ s \xrightarrow{a} s'\}$. For a set of states S, we have $a(S) = \{a(s) \mid s \in S\}$ and $en(S) = \{en(s) \mid s \in S\}$. With $en_T(s)$ we denote the set of enabled transitions at s, that is $en_T(s) = \{(s, a, s') \mid a \in en(s) \wedge s' \in a(s)\}$. A

state s is called *deadlock* iff $en(s) = \emptyset$. We say L is *deterministic* iff $\forall s \in S.\ |en_T(s)| \leq 1$. [1] LTS L is called *single-image* iff $\forall a \in A, s \in S.\ |a(s)| \leq 1$. Note that being single-image does *not* entail determinacy. For example, the LTS of figure 4.1 is single-image, but it is not deterministic. As a convention, in graphical representation of LTSs the initial state is distinguished by having an unlabelled arrow pointing to it that does not have any source. Let $\rightarrow^+$ be the transitive closure of $\cup_{a \in A} \stackrel{a}{\rightarrow}$. An LTS is *acyclic* iff $\neg\exists s \in S.\ (s, s) \in \rightarrow^+$. Obviously, finiteness does not entail acyclicity.

A *trace*, or execution, rooted at state s is a sequence of actions $\alpha = \alpha_1 \cdot \alpha_2 \cdots$ such that $\alpha_i \in en(\alpha_{i-1}(\cdots \alpha_1(s)))$ for $i \geq 1$. We write $\pi(s)$, with $s \in S$, for the set of traces that emanate from s. The empty trace, denoted by ϵ, belongs to $\pi(s)$ for any s. As an example, in the LTS of figure 4.1 we have $\pi(s_0) = \{\epsilon, a, b\}$. A trace α is said to *be in* L if $\alpha \in \pi(s_0)$.

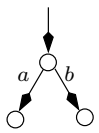


Figure 4.1: *A simple LTS*

To a trace $\alpha = \alpha_1 \cdot \alpha_2 \cdots$, we attribute a sequence of states $s^\alpha = s_0^\alpha \cdot s_1^\alpha \cdots$ and a sequence of transitions $t^\alpha = t_1^\alpha \cdot t_2^\alpha \cdots$, such that $s_0^\alpha = s_0$, $s_i^\alpha \in \alpha_i(s_{i-1}^\alpha)$ and $t_i^\alpha = (s_{i-1}^\alpha, \alpha_i, s_i^\alpha)$. The sequence of *ready transitions* associated to trace α, denoted by T^α, is defined as $en_T(s_0^\alpha), en_T(s_1^\alpha), \cdots$. Clearly $t_i^\alpha \in T_i^\alpha$. Note that in single-image LTSs, s^α, t^α and T^α are unique for each α. A trace α is *maximal* if it is either an infinite sequence or it reaches a deadlock state, i.e. $\exists i.\ en(s_i^\alpha) = \emptyset$.

Fairness constraints are used to avoid unrealistic model behaviours which do not reflect realistic executions of the system. [2] Executions which violate fairness constraints are usually omitted from further (formal) analysis. For example, consider a communication channel between A and B, which may lose messages but is never cut (such a channel can be used to implement resilient channels of § 2.2.2). An execution of the system in which A sends an infinite number of messages but only a finite subset of them reaches B is deemed unfair.

Below, we define a fairness constraint, called $\mathcal{F}_0$, stating that each transition which becomes available infinitely often, is realised infinitely often. This constraint is often necessary when analysing nondeterministic asynchronous systems. Note that even when system components are deterministic per se, nondeterminism in the model is typically unavoidable when they are executed in parallel. Our definition of $\mathcal{F}_0$-fair traces coincides with the concept of *fair choice from states* in [QS83] and the strong notion of fairness [3] in [Fra86].

[1] We note that *deterministic* is an overloaded adjective, and our use can be substantially different from what the reader might expect from "deterministic" systems.

[2] Fairness constraints on LTSs should not be confused with fairness in exchange protocols as defined in § 2.2.

[3] This condition states that $\forall \theta \in Tr.\ F^\infty\ \mathrm{enabled}(\theta) \Rightarrow F^\infty\ \mathrm{executed}(\theta)$.

4.1. DEFINITION. *Let α be a trace in a finite LTS $L = (S, s_0, A, Tr)$. We say α is $\mathcal{F}_0$-fair iff*

- *if α is finite, namely $\alpha = \alpha_1 \cdots \alpha_n$, then α is maximal (i.e. $T_n^\alpha = \emptyset$).*
- *if α is infinite, namely $\alpha = a_1 \cdot a_2 \cdots$, then for each $\theta \in Tr$, if $\{i \mid \theta \in T_i^\alpha\}$ is infinite, then so is $\{i \mid \theta = t_i^\alpha\}$.*

When a trace α corresponds to multiple traces of states and transitions in L, then for α to be fair, at least one of these sequences should satisfy the conditions of definition 4.1. This is motivated by considering fairness as a property of traces, in which transitions play only a supporting role. We recall that in single-image LTSs, each trace has a unique sequence of states (and hence transitions) associated to it.

Various fairness constraints have been introduced in the literature and the relations among them have been studied, e.g. see [Fra86, VV06]. In fact $\mathcal{F}_0$ is among the strongest fairness constraints that can be devised for finite state systems, cf. [Fra86]. We have defined $\mathcal{F}_0$-fair traces at the level of transitions, as opposed to actions, to more easily reject certain traces as unfair. Consider, for example, the infinite trace $\alpha = a \cdot b \cdot a \cdot b \cdots$ in $(\{s_0, s_1, s_2\}, s_0, \{a, b\}, \{s_0 \xrightarrow{a} s_1, s_1 \xrightarrow{a} s_2, s_1 \xrightarrow{b} s_0\})$. If definition 4.1 was based on actions, not transitions, then the trace α would have been considered fair in this LTS, despite the fact that it visits s_1 infinitely often but from there never performs a to move to s_2. In fact, the soundness of our main results in § 5 depends on the $\mathcal{F}_0$ notion of fairness, in which if a transition is infinitely often enabled, then it should be infinitely often realised.

To each μCRL specification we associate an LTS in which states represent process terms and transitions are labelled with the actions that can be performed by that process term. The semantics of μCRL describes how such LTSs are composed [GP95]. For instance, the LTS associated to the process $X = \Sigma_{b:Bool} send(b).X$ with $Act = \{send\}$ is shown in figure 4.2. See also § 7.2.1.

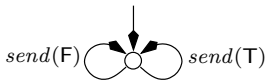


Figure 4.2: *The LTS describing $X = \Sigma_{b:Bool} send(b).X$*

To efficiently generate and manipulate state spaces associated with processes, the μCRL language comes with the benefit of a strong tool support. We refer to [Wou01, BFG+01, GL02b, BCL+07, Fok07, BLPW07] for discussions on functionalities and internal details of the μCRL tool-set.

4.1.2 Specifying security protocols in μCRL: An example

Below, as an example, we describe how to specify a simple authentication protocol (used in [Tan96a] to demonstrate reflection attacks) and the $\mathcal{DY}$ intruder, along with its data ma-

nipulation abilities, in μCRL. The μCRL language and tool-set have been used in formal specification and verification of various security protocols. See, for instance, [Pan02, CT04, CCT05, COPT07, TKJ07, CPT07]. In this example, we introduce several patterns and concepts to which we will return in the rest of the thesis. Our specification in many ways follows the approach of Clarke et al. [CJM98] and Paulson [Pau98].

The protocol of our example can informally be described as follows. It is assumed that A and B share a secret symmetric key K_{AB}, and r_A and r_B are nonces.

$$\begin{aligned} &A \to B : A, r_A \\ &B \to A : r_B, \{r_A\}^s_{K_{AB}} \\ &A \to B : \{r_B\}^s_{K_{AB}} \end{aligned}$$

The protocol is intended for mutual authentication. However, it is susceptible to the following reflection attack.

$$\begin{aligned} &Z(A) \to B : A, r_Z && \text{\#1} \\ &B \to Z(A) : r_B, \{r_Z\}^s_{K_{AB}} && \text{\#1} \\ &Z(A) \to B : A, r_B && \text{\#2} \\ &B \to Z(A) : r'_B, \{r_B\}^s_{K_{AB}} && \text{\#2} \\ &Z(A) \to B : \{r_B\}^s_{K_{AB}} && \text{\#1} \end{aligned}$$

In this attack scenario, the intruder Z, impersonating A, opens two sessions with B and solves B's challenge in the first session (i.e. "please compute $\{r_B\}^s_{K_{AB}}$ if you are really A") using B in the second session. The attack is feasible only if B permits concurrent sessions.

We consider a small system consisting of one honest party, B, and the $\mathcal{DY}$ attacker who tries to impersonate A. We thus let $\mathcal{P} = \{A, B, \mathcal{DY}\}$. Let $\mathcal{T} = \{t_1, \cdots, t_\ell\}$ be a finite set of atomic terms, and $\mathcal{P} \subset \mathcal{T}$. For simplicity, below, we omit asymmetric encryption primitives (cf. § 2.1.1). First, we define the sort of messages exchanged in the protocol. With this definition, an induction principle comes for free.

$$\begin{aligned} \textbf{sort} \quad & \mathit{Msg} \\ \textbf{func} \quad & t_1, \cdots, t_\ell :\rightarrow \mathit{Msg} \\ & \cdot, \cdot : \mathit{Msg} \times \mathit{Msg} \to \mathit{Msg} \\ & h : \mathit{Msg} \to \mathit{Msg} \\ & \{\cdot\}^s_\cdot : \mathit{Msg} \times \mathit{Msg} \to \mathit{Msg} \end{aligned}$$

The constants (elements of $\mathcal{T}$) and the message constructors (pairing [4], hashing and encryption) constitute the constructors for the Msg data type. These are defined under the keyword **func** in μCRL, while the keyword **map** is used when defining non-constructor functions associated to data types. Next, term rewriting, denoted by $\Rightarrow$, [5] is used to define equality for Msg. The μCRL tool-set acts as a term rewriter and assumes that the rewrite rules which

[4] To improve readability, we sometimes put parentheses around pairs of messages.

[5] To denote rewriting we use $\Rightarrow$, while logical implication is denoted by $\Longrightarrow$.

specify data types constitute a weakly terminating and confluent system [GP95]. This guarantees that data equivalence between closed terms is decidable.

Semantic equality in ideal cryptography collapses to syntactic identity, i.e. two messages are equal iff they are identical. The elements of Msg, therefore, constitute a *free algebra*, meaning that no relations (except identity) on Msg are assumed. This abstraction is restrictive, as it does not capture the algebraic properties of the cryptographic primitives used in protocols, cf. § 2.1.1. In fact, extending this algebra even with an explicit decryption operator (along with the corresponding equality relation) gives strictly more power to the attacker. The latter shortcoming can be circumvented by, intuitively, requiring protocol specifications to not have any encryption operator applied on a variable by itself, see [Mil03, LM05].

$$
\begin{array}{ll}
\textbf{map} & \cdot = \cdot : Msg \times Msg \to Bool \\
\textbf{var} & m, m', m_1, m_1' : Msg \\
\textbf{rew} & t_1 = t_1 \Rightarrow \mathsf{T} \quad \cdots \quad t_\ell = t_\ell \Rightarrow \mathsf{T} \\
 & (m, m') = (m_1, m_1') \Rightarrow m = m_1 \wedge m' = m_1' \\
 & h(m) = h(m') \Rightarrow m = m' \\
 & \{m\}^s_{m'} = \{m_1\}^s_{m_1'} \Rightarrow m = m_1 \wedge m' = m_1'
\end{array}
$$

Inequalities also have to be defined. This is because the above rewrite rules for equality lead to unspecified states. For example, these do not determine whether $t_1 = h(t_2)$ holds or not, simply because the left-hand side of no rule matches this. Inequalities are defined in a natural way: Different kinds of messages are distinct, with $t_i = t_j \Rightarrow \mathsf{F}$ for $i \neq j$, $h(m) = \{m_1\}^s_{m_2} \Rightarrow \mathsf{F}$, and so forth.

This completes our specification of the Msg data type. Next, we specify the attacker's deduction abilities. To represent $\mathcal{DY}$'s knowledge, we use a set of messages. In practice, often ordered lists, instead of sets, are used. This gives a normal form for message sets and,

thus, avoids storing the same object multiple times in different representations.

$$
\begin{array}{ll}
\textbf{sort} & \mathit{Set} \\
\textbf{func} & \emptyset :\rightarrow \mathit{Set} \\
 & \varsigma(\cdot,\cdot) : \mathit{Msg} \times \mathit{Set} \rightarrow \mathit{Set} \\
\textbf{map} & \cdot \in \cdot : \mathit{Msg} \times \mathit{Set} \rightarrow \mathit{Bool} \\
 & \mathit{add}(\cdot,\cdot) : \mathit{Msg} \times \mathit{Set} \rightarrow \mathit{Set} \\
 & \cdot \cup \cdot : \mathit{Set} \times \mathit{Set} \rightarrow \mathit{Set} \\
 & \cdot \subseteq \cdot : \mathit{Set} \times \mathit{Set} \rightarrow \mathit{Bool} \\
 & \cdot = \cdot : \mathit{Set} \times \mathit{Set} \rightarrow \mathit{Bool} \\
\textbf{var} & m : \mathit{Msg} \\
 & S, S' : \mathit{Set} \\
\textbf{rew} & m \in \emptyset \Rightarrow \mathsf{F} \\
 & m \in \varsigma(m', S) \Rightarrow (m = m') \vee (m \in S) \\
 & \mathit{add}(m, S) \Rightarrow \mathit{if}(m \in S, S, \varsigma(m, S)) \\
 & \emptyset \cup S' \Rightarrow S' \\
 & \varsigma(m, S) \cup S' \Rightarrow \mathit{add}(m, S \cup S') \\
 & \emptyset \subseteq S' \Rightarrow \mathsf{T} \\
 & \varsigma(m, S) \subseteq S' \Rightarrow m \in S' \wedge S \subseteq S' \\
 & S = S' \Rightarrow S \subseteq S' \wedge S' \subseteq S
\end{array}
$$

In this specification, we define $\mathit{if} : \mathit{Bool} \times \mathit{Set} \times \mathit{Set} \rightarrow \mathit{Set}$ as $\mathit{if}(\mathsf{T}, S_1, S_2) \Rightarrow S_1$ and $\mathit{if}(\mathsf{F}, S_1, S_2) \Rightarrow S_2$.

Let Γ represent $\mathcal{DY}$'s knowledge set. To capture the deduction rules of definition 2.2, we follow the approach of [CJM98] and [Pau98]. [6] We first maximally decompose Γ using $\mathcal{DR}$ rules, and then define a synthesise function *synth* which only implements the $\mathcal{CR}$ rules of definition 2.2 on the resulting decomposed set. This entire procedure corresponds to computing $\mathsf{synth}(\mathsf{analz}(\Gamma))$ in the terminology of [Pau98].

The $\mathcal{DR}$ rules are implemented in three phases. In the first phase, encrypted messages are not decrypted:

$$
\begin{array}{ll}
\textbf{map} & \mathit{decomp} : \mathit{Msg} \rightarrow \mathit{Set} \\
\textbf{var} & m, m' : \mathit{Msg} \\
 & S : \mathit{Set} \\
\textbf{rew} & \mathit{decomp}(t_1) \Rightarrow \varsigma(t_1, \emptyset) \quad \cdots \quad \mathit{decomp}(t_\ell) \Rightarrow \varsigma(t_\ell, \emptyset) \\
 & \mathit{decomp}((m, m')) \Rightarrow \mathit{decomp}(m) \cup \mathit{decomp}(m') \\
 & \mathit{decomp}(h(m)) \Rightarrow \varsigma(h(m), \emptyset) \\
 & \mathit{decomp}(\{m\}^s_{m'}) \Rightarrow \varsigma(\{m\}^s_{m'}, \emptyset)
\end{array}
$$

The *decomp* function is extended to sets of messages in the natural way. In the second phase, the function *decrypt* extracts the messages which are encrypted with those keys that can

[6] The expositions of [CJM98, Pau98] are different from ours as they assume that the keys used for symmetric encryption have a distinct type and cannot be an arbitrary message (i.e. atomic key assumption).

immediately be synthesised (the function $synth$ is defined later). $decrypt$ is to be applied as $decrypt(decomp(\Gamma), decomp(\Gamma))$.

$$
\begin{array}{ll}
\textbf{map} & decrypt\ : Set \times Set \rightarrow Set \\
\textbf{var} & S, S'\ : Set \\
 & m, m' : Msg \\
\textbf{rew} & decrypt(\emptyset, S) \Rightarrow \emptyset \\
 & decrypt(\varsigma(t_1, S), S') \Rightarrow add(t_1, decrypt(S, S')) \\
 & \vdots \\
 & decrypt(\varsigma(t_\ell, S), S') \Rightarrow add(t_\ell, decrypt(S, S')) \\
 & decrypt(\varsigma(h(m), S), S') \Rightarrow add(h(m), decrypt(S, S')) \\
 & decrypt(\{m\}^s_{m'}, S), S') \Rightarrow if(synth(m', S'), \\
 & \qquad\qquad\qquad\qquad decomp(m) \cup decrypt(S, S'), \\
 & \qquad\qquad\qquad\qquad add(\{m\}^s_{m'}, decrypt(S, S')))
\end{array}
$$

In case (parts of) keys are encrypted using other keys, the $decrypt$ function needs to be iterated to capture all possible decryptions. For instance, for $\Gamma = \{t_1, \{t_2\}^s_{t_1}, \{t_3\}^s_{t_2}\}$, we expect that $\Gamma \vdash t_3$, while $decrypt(\Gamma, \Gamma) \Rightarrow \{t_1, t_2, \{t_3\}^s_{t_2}\}$. We name the last phase of decomposition as $decrypt^*$:

$$
\begin{array}{ll}
\textbf{map} & decrypt^*\ : Set \rightarrow Set \\
\textbf{var} & S : Set \\
\textbf{rew} & decrypt^*(S) \Rightarrow if(S = decrypt(S, S), \\
 & \qquad\qquad\qquad\qquad S, \\
 & \qquad\qquad\qquad\qquad decrypt^*(decrypt(S, S)))
\end{array}
$$

Now, we turn to the $synth$ function which implements the $\mathcal{CR}$ rules of definition 2.2.

$$
\begin{array}{ll}
\textbf{map} & synth\ : Msg \times Set \rightarrow Bool \\
\textbf{var} & m, m' : Msg \\
 & \Gamma : Set \\
 & p, p' : \mathcal{P} \\
\textbf{rew} & synth(t_1, \Gamma) \Rightarrow t_1 \in \Gamma \quad \cdots \quad synth(t_\ell, \Gamma) \Rightarrow t_\ell \in \Gamma \\
 & synth((m, m'), \Gamma) \Rightarrow synth(m, \Gamma) \wedge synth(m', \Gamma) \\
 & synth(h(m), \Gamma) \Rightarrow h(m) \in \Gamma \vee synth(m, \Gamma) \\
 & synth(\{m\}^s_{m'}, \Gamma) \Rightarrow \{m\}^s_{m'} \in \Gamma \vee (synth(m, \Gamma) \wedge synth(m', \Gamma))
\end{array}
$$

To decide $\Gamma \vdash m$, we check whether $synth(m, decrypt^*(decomp(\Gamma))) \Rightarrow \mathsf{T}$ or not. It has been proved in [CJM98] that this procedure is terminating and indeed implements the deduction rules of definition 2.2, while assuming that only atomic keys are used. A similar approach is taken in [Bol96, Pau98]. Below, we informally contend that for any finite Γ, possibly containing messages encrypted with composed keys, $decrypt^*(decomp(\Gamma))$ terminates after a finite number of iterations.

We first define a function to count the number of encrypted messages appearing in a set of messages: Define $ne : Msg \to \mathbb{N}$ as $\forall t \in \mathcal{T}.\ ne(t) = 0$, $ne(h(m)) = 0$, $ne(\{m\}^s_{m'}) = ne(m) + 1$ and $ne(m, m') = ne(m) + ne(m')$. For a set of messages S, let $ne(S) = \Sigma_{m \in S} ne(m)$ and $ne(\emptyset) = 0$. The claim is that, for a finite set S such that $decomp(S) = S$, the function $decrypt^*(S)$ terminates after at most $ne(S) + 1$ iterations. Define $S^i = decrypt(S^{i-1}, S^{i-1})$ and $S^0 = S$. Note that $S^i \neq S^{i-1}$ iff an encryption appearing in S^{i-1} is decrypted in S^i. Since the number of encryptions appearing in S is $ne(S)$, after $ne(S)$ iterations all the encryptions are decrypted or cannot be decrypted at all using the terms appearing in $S^{ne(S)}$. Therefore, in the $ne(S) + 1^{th}$ iteration, $decrypt^*$ reaches a fixed point, and thus terminates. [7] This completes our data specification part.

Below, we specify the participants of our simple authentication protocol. In this, we follow the approach of Woo and Lam [WL93]: "The specification in fact describes ... distinct *local* protocols, each of which specifies the actions of one of the participating principals". Our specification of the $\mathcal{DY}$ process is generic, while protocol participants are assigned with protocol specific code, as is shown below.

First, we give the definitions of action labels.

$$\begin{array}{ll} \textbf{act} & send_B, recv_B, send_I, recv_I, \mathbf{send}_B, \mathbf{recv}_B : Msg \\ & auth_B \\ \textbf{comm} & send_B | recv_I = \mathbf{send}_B \\ & send_I | recv_B = \mathbf{recv}_B \end{array}$$

Intuitively, Bob uses $send_B(m)$ and $recv_B(m)$ actions for sending and receiving $m \in Msg$ data term over public channels. Moreover, $send_B$ and $recv_B$ may synchronise, respectively, with $recv_I$ and $send_I$ intruder actions and the resulting actions are distinguished by their bold font. The action $auth_B$ is used to denote the point at which B believes (or claims) that he has authenticated A. We assume that there is no A in the system and, instead, the $\mathcal{DY}$ intruder wants to convince B that he is A. The specification for B goes as follows.

$$\begin{array}{lll} \text{Bob } (r_B : Msg) & = \Sigma_{X:Msg} & (recv_B(X) \lhd fst(X) \in \mathcal{P} \rhd \delta) \cdot \\ & & send_B(r_B, \{snd(X)\}^s_{K(fst(X))}) \cdot \\ & & recv_B(\{r_B\}^s_{K(fst(X))}) \cdot \\ & & auth_B \cdot \delta \end{array}$$

Here fst and snd, when applied to a message pair (m, m'), return m and m', respectively, and when applied to other messages $fst(m) = snd(m) = m$. The function K characterises the set of keys that Bob shares with other participants: $K(A)$ is the key that B shares with A, namely $K(A) \Rightarrow K_{AB}$, etc.

[7] As computing $decrypt^*$ can be non-terminating (e.g. using inner-most strategy), a special treatment of the *if* structure is needed when generating LTSs using the μCRL tool-set. For this purpose, the lazy rewriter jitty should be instantiated as the rewrite engine, see [Pol01].

Below we specify the $\mathcal{DY}$ attacker process:

$$\begin{aligned}\mathcal{DY}(\Gamma : Set) = \quad & \Sigma_{m \in Msg} \\ & \qquad recv_I(m) \cdot \mathcal{DY}(decrypt^*(decomp(m) \cup \Gamma)) \\ & + \\ & \Sigma_{m \in Msg} \\ & \qquad send_I(m) \cdot \mathcal{DY}(\Gamma) \lhd synth(m, \Gamma) \rhd \delta\end{aligned}$$

The $\mathcal{DY}$ attacker, as discussed before, simply intercepts all messages, adds them to its knowledge set Γ, and can send messages if he can synthesise them from Γ. Note that when adding new messages to Γ, Γ is maximally decomposed and decrypted. Checking $synth(m, \Gamma)$ is thus equal to checking $synth(m, decrypt^*(decomp(\Gamma)))$, cf. [CJM98]. A benefit of decomposing, before adding messages, is that Γ does not grow unnecessarily: In our specification, receiving, e.g., the message $t_1, \{t_2\}^s_{t_1}$ when $\Gamma = \{t_1, t_2\}$ does not change Γ. [8] This can further be optimised by adding only those messages to Γ that cannot be synthesised using Γ, namely to define add alternatively as $add(m, S) = if(synth(m, S), S, \varsigma(m, S))$. As we see in the following, keeping Γ minimal can reduce the size of the resulting LTS, which is advantageous for model checking purposes.

To initiate the system, we allow Bob to execute two parallel sessions. As mentioned earlier, this is necessary for the reflection attack of [Tan96a] to work. The initial knowledge of $\mathcal{DY}$ is defined as $\Gamma_0 = \varsigma(\mathcal{DY}, \varsigma(A, \varsigma(r_A, \emptyset)))$. We thus have

$$\textbf{init} \quad L = \partial_{\{send_B, recv_B, send_I, recv_I\}}(\text{Bob}(r_{B_1}) \| \text{Bob}(r_{B_2}) \| \mathcal{DY}(\Gamma_0))$$

To check the authentication property of the protocol, we need to test if the action $auth_B$ ever occurs in L. This is because $auth_B$ shows that B believes he is talking to A (in this instantiation), and as A is absent in L, it must be $\mathcal{DY}$ who has successfully impersonated A.

For a formal treatment of how the LTS corresponding to L is constructed, we refer to the operational semantics of μCRL [GP95]. See also § 7.2.1. In L, intuitively, each state corresponds to the Cartesian product of the states of the two Bob processes and the state of the $\mathcal{DY}$ processes. The state of (each) process Bob is uniquely determined by the process remaining to be executed by Bob. The state of the $\mathcal{DY}$ process is characterised by Γ. Therefore, syntactically different Γ sets would lead to different intruder states. For example, $\Gamma_1 = \varsigma(A, \varsigma(r_A, \emptyset))$ and $\Gamma_2 = \varsigma(r_A, \varsigma(A, \emptyset))$, although semantically equal, are syntactically different, and would thus represent "different" intruder states. A simple way to avoid such superfluous states is to implement Γ as an ordered set (see [CT04, CCT05, TKJ07] for such specifications).

We also note that the LTS associated to L is in general infinite. This is because the $\mathcal{DY}$ process can generate an infinite number of messages and pass them to B as, e.g., X. Note that Bob generates only data terms that belong to finite data domains. If Bob could nondeterministically choose, e.g., keys from an infinite domain, then L would become infinitely

[8] This optimisation method has been used in [RG97].

branching, regardless of the intruder's abilities. To keep L finite, we could, for instance, limit the range of terms that B accepts as X. For example, by letting $recv_B(X) \lhd fst(X) \in \mathcal{P} \wedge snd(X) \in \{r_A, r_{B_1}, r_{B_2}\} \rhd \delta$, the synchronisation $send_I|recv_B$ automatically ensures that the messages not acceptable by B are not sent by $\mathcal{DY}$, hence making L finite. This abstraction is tenable only if B can detect the type of each part of the received messages, e.g., via type tags (cf. [HLS03]). Type flaw attacks are thus obscured in this *type-correct* construction. [9] This approach is widely used in finite state model checking for security protocols (e.g. [MMS97, LBL+99, RSG+00, KR01, CCT05]). Other solutions to keep L's state space finite include restricting the depth of $\mathcal{DY}$'s deduction proofs (e.g. [CJM98]) or imposing a bound on message lengths.

We remark that $\mathcal{DY}$'s ability to generate fresh atomic terms does not need to be limited when assuming type-correct messages in L: It can generate and use fresh atomic terms, one by one. This is due to the following features of process Bob: It is acyclic and non-replicating, and, moreover, it can infer the form of X. If Bob was cyclic or replicating, then he could receive a fresh value from the attacker in each round, hence making L infinite. In case X was a term encrypted for A, which Bob could not parse, then the type-correct assumption would not entail finite L, simply because the data type Msg has an infinite domain.

This example completes our introduction to process algebraic specification of cryptographic protocols. For μCRL specifications of larger security protocols see § 6.

4.2 Modal logics

To formulate (un)desired properties of protocols, we use the alternation-free fragment of the regular μ-calculus. This choice gives us enough expressiveness (properties of the security protocols can be expressed in this logic, fairness constraints can naturally be encoded as part of logic, etc), and it can efficiently be model checked. Below, we shortly introduce this logic, while its complete syntax and semantics can be found in [MS03]. For a general introduction to μ-calculus we refer to [BS01].

Formulae of alternation-free regular μ-calculus are interpreted on LTSs and can be model checked in polynomial time on finite systems. These consist of *state formulae*, which may contain *regular formulae*. Regular formulae describe sets of traces [10] and are built upon *action formulae* and the standard regular expression operators. We use '$\cdot$', '$\vee$', '$\neg$' and '*' for concatenation, choice, complement and transitive-reflexive closure, respectively, of regular formulae. For example, the trace $a{\cdot}a{\cdot}b{\cdot}c{\cdot}c$ belongs to the regular trace formula $a^*{\cdot}\neg(a{\vee}c){\cdot}c^*$.

[9] The vocabulary type-correct is borrowed from [RG97]. A similar concept is called *strong typing abstraction* in [HLS03]: "Most protocol analysis techniques adopt the strong typing abstraction, where all messages considered in the analysis are assumed to be well-typed. This corresponds to an assumption that all agents can 'magically' tell the true types of messages.".

[10] Here we use "trace" to refer to any sequence of actions, cf. § 4.1.1.

A state formula, characterising a set of states, is built upon propositional variables, standard Boolean operators, the possibility modal operator $\langle\ \rangle$, the necessity modal operator $[\ ]$, and the minimal and maximal fixed point operators μ and ν. The symbols F and T are used in both action formulae and state formulae. In action formulae they represent *no action* and *any action* and in state formulae they denote the empty set and the entire state space, respectively. The wild-card action parameter '_' represents any parameter of an action. Given LTS L and state s in L, we write $s \models \phi$ to denote that state s satisfies formula ϕ. If s is the initial state of L, we may also write $L \models \phi$.

4.1. EXAMPLE. *Let L be an LTS and s be a state in L. When R is a regular formula, s satisfies $\langle R\rangle\mathsf{T}$ iff there exists a trace α in L, emanating from s, such that $\alpha \in R$. Similarly, $s \models [R]\mathsf{F}$ iff, for any trace α that emanates from s, we have $\alpha \notin R$. Going back to the example of § 4.1.2, to verify authentication, we can check if $L \models [\mathsf{T}^*.auth_B]\mathsf{F}$ holds.*

A state satisfies $\mu X.\ f$ iff it belongs to the minimal solution of the fixed point equation $X = f(X)$, with f being a monotonic state formula and X a variable (correspondingly defined for the greatest fixed point operator ν). Here, the formula f represents a mapping from sets of states to sets of states: A set S of states is mapped to those states where f holds, under the assumption that the recursion variable X evaluates to T for states inside S, and to F for states outside S. For f being monotonic, X may only occur under an even number of negations in f. The image of S, under the mapping f, is denoted by $FIX(f, X{=}S)$ (here we borrow our notations from § 7.3 of [Fok07]). To compute $\mu X.\ f$, initially we let $X = \emptyset$ (with set inclusion being the partial ordering on sets of states, the least and the greatest element are the empty set and the set of all states, respectively). Next, we repeatedly compute $S_{i+1} = FIX(f, X{=}S_i)$ for $i \geq 0$. This iterative computation reaches a fixed point when $S_i = S_{i+1}$, for some i. The set S_i is then said to satisfy $\mu X.\ f$. [11]

4.2. EXAMPLE. *The pattern $\vartheta = \mu X.\ \langle\mathsf{T}\rangle\mathsf{T} \wedge [\neg a]X$, with $a \in Act$, captures inevitable reachability of a. Since μ is the minimal fixed point operator, we initially let $X = \emptyset$. After the first iteration, X contains all the states that can perform the action a, and from which no other action can be performed. After the n^{th} iteration, X contains all states from which every trace contains within n transitions the action a. So as a fixed point formula, $\mu X.(\langle\mathsf{T}\rangle\,\mathsf{T} \wedge [\neg a]\,X)$ holds in those states from which every maximal trace eventually contains the action a.*

Alternation-free formulae, intuitively, avoid alternation of μ and ν operators. These formulae can efficiently be model checked in the CADP verification tool-set [FGK+96]. The following example shows how *fair reachability* can be modelled in this logic.

4.3. EXAMPLE. *Let L be the LTS corresponding to process $p = a \cdot p + b.\delta$, where $a, b \in Act$, demonstrated in figure 4.3. Note that b is not inevitably reachable from s_0, because of the*

[11] Intuitively, μ corresponds to a finite number of iterations through the recursion, while ν insists on an infinite number of iterations.

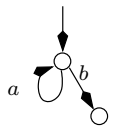


Figure 4.3: *The LTS of example 4.3*

cycle. However, if we confine our attention to fair traces (see definition 4.1), then b *is indeed (fairly) inevitably reachable from* s_0*. Thus,* $L \not\models \phi_1$*, but* $L \models \phi_2$*.*

$$\phi_1 = \mu X.\, \langle \mathsf{T} \rangle \mathsf{T} \wedge [\neg b] X$$
$$\phi_2 = [(\neg b)^*] \langle \mathsf{T}^* \cdot b \rangle \mathsf{T}$$

Intuitively, ϕ_2 *states that while* b *has not happened, there is always a path to* b*. Therefore, if fair traces are considered, this path(es) should eventually be taken, hence* b *should eventually be reached.*

Another modal logic which is used in the thesis is an action-based variant of the linear temporal logic (LTL) [Pnu77]. This logic, which is subsumed by regular alternation-free μ-calculus, is described in § 7.2.3.

4.3 Formal verification

We use finite state model checking to verify finite models of security protocols, expressed in μCRL, against their requirements, expressed in alternation-free regular μ-calculus. As we merely make use of existing tools and techniques, details of model checking algorithms are not discussed here (see, e.g., [CGP00, MS03]). Instead, we motivate this choice for validation of FE protocols.

As mentioned in § 2.1.1, checking even simple properties such as reachability in security protocols, in presence of $\mathcal{DY}$, is in general undecidable. This makes automated security proofs generally out of reach. Semi-automated methods can be used to formally reason about security protocols, see, e.g., [Bol96, DS97, Pau98, BP01, AB03, ES05a]. These, nonetheless, require non-negligible human intervention to guide the proofs.

Systems describing the behaviour of security protocols in the $\mathcal{DY}$ model tend to be infinite. However, restricting participants behaviours guarantees decidability for many security related questions in such infinite systems. In particular, with acyclic participant specifications, assuming a bounded number of protocol sessions makes reachability [RT01, MS01] and certain branching properties [KKW05, KK05], which comprise requirements of FE protocols, decidable, although the system remains infinite.

In FE protocols, a natural way to model TTPs (and in some cases, content vendors as well, see [CT04]) is to represent them as cyclic [12] processes: The TTP receives a resolve request, processes the request, updates its database, and recurs. With cyclic participant processes, reachability is not decidable in security protocols [HS04]. In fact, even if these cycles are idle (i.e. do not consume or generate messages), then checking *strategy properties* is undecidable [KKT07]. Strategy properties, intuitively, concern strategies of one process, e.g. the attacker, to reach a goal in spite of a coalition of the other processes in a protocol. These properties encode in particular the requirements of FE protocols [KR01]. Therefore, automatic security proofs for FE protocols is unattainable, if we stay faithful to cyclic processes in the model. We can however confine to finite cyclic state spaces, the pros and cons of which are discussed below.

To enforce finite state spaces for security protocols, the infinitely branching behaviour of $\mathcal{DY}$ [13] has to be limited as well (besides assuming a finite number of protocol rounds). For this purpose, it is usually assumed that the messages are type-correct (see § 4.1.2). Furthermore, we need to assume that the number of fresh atomic terms generated by (cyclic) participants and the intruder is bounded. [14] These assumptions degrade the impact of finite state security analyses, compared to infinite state decision algorithms, as it, e.g., obscures type flaw attacks.

Nonetheless, no decision procedure exists for FE properties when considering cyclic participant specifications. Therefore, using existing decision algorithms for FE requirements is penalised by incomplete participant specification. Conversely, opting for finite state analysis has the benefit of natural and complete specification of participants, but does not allow the attacker to generate an unbounded number of fresh atomic terms, and lacks the capability of detecting type flaws. Here, we favour the second approach. A thorough analysis would incorporate both these methods to get a more detailed perspective of protocols.

It is also worth mentioning that finite state analysis can usually benefit from an expressive specification language. For instance see the specification of § 4.1.2 and § 6, which can readily be translated into executable programs. Whereas most existing decidability results for security protocols hinge on the assumption that protocol participants are specified as very simple rewrite rules. To model, e.g., the TTP of a FE protocol in such a rewrite framework seems rather awkward, and certainly far from how it would be implemented in practice, e.g. see the specification of TTPs in [KKW05, KKT07].

Below, we comment on the general use of formal methods in verifying security protocols, mostly from a critical point of view. Benefits of formal methods in the security of systems are abundantly described in the literature, see, e.g., [Mea03].

Security proofs derived in the Dolev-Yao model have to be taken with great prudence,

[12] The class of cyclic processes is not comparable with the class of *recursive* processes à la [Pau97].

[13] Recall that $\Gamma \neq \emptyset$ implies that $\{m \mid \Gamma \vdash m\}$ is infinite.

[14] A similar restriction is applied in [CV02, CKR$^+$03] where the authors consider a setting in which some participants can engage in an unbounded number of sessions, while the others cannot. Those who can participate in an unbounded number of sessions can only generate a bounded number of fresh atomic terms.

as these generally abstract away cryptographic details of the protocols (for more on this see § 2.1.1), often consider only a bounded number of protocol sessions, and study protocols in isolation (cf. [KSW98]). "Security models and formal methods", in general, "do not establish security. Systems are hacked outside the models' assumptions", as Dorothy Denning put it [Den99]. See also [Gol06a, Kob07] on the notion of "proof" in security proofs.

Formal techniques can however serve well as bug hunters. Security flaws are almost constantly reported and not only serve as a justification for yet-another formal technique or tool which has found them, but are also counted as valuable contributions of the formal methods community to the security community. These have however not been as warmly embraced by security researchers. Perhaps partly because, in this game, protocol designers are the community under attack, and partly because many flaws found by formal methods are not considered as significant attacks, e.g. they study a protocol out of the context for which it has been designed, cf. [Pan00, PG06]. Since formal methods in general do not provide security proofs, reporting some witness of insecurity gains superficial importance, as otherwise, seemingly, formal verifications are of no use. Reported security flaws should thus be accepted with just as much of circumspection as security proofs.

One could however rightly argue that these flaws refine the conditions under which a protocol behaves as expected. This takes us to the third use of formal methods in security engineering. Formal methods have excelled in precisely defining security problems and providing insights into their solutions. Notable examples are formalisation of various non-interference concepts [FG01], classifying security requirements [Low97, Mea04] and guidelines in designing authentication protocols [KG91, BFM03].

Chapter 5

Verifying liveness in security protocols

Death is forever. It doesn't matter when it begins.

Emir Kusturica's *Crna mačka, beli mačor* (1998)

5.1 Introduction

Liveness aspects of security protocols, stipulating that some desired event will inevitably occur, have only recently found a role on the scene with the requirements emerging from electronic commerce applications. Fair payment, certified email, non-repudiation and electronic contract signing protocols are examples of relevant applications (see § 2.2). These protocols all aim at a liveness requirement, namely termination of the protocol in a fair state.

In general, liveness requirements of security protocols cannot be achieved unless messages are eventually delivered over (some) communication channels. This is a result of the impossibility of reaching agreement in the presence of faulty channels, cf. the generals paradox [Gra78], described in § 2.2.1.

In modelling security protocols, following the approach of Dolev and Yao [DY83], the communication media are assumed to be under complete control of the intruder. The $\mathcal{DY}$ intruder can in particular destroy transmitted messages. For liveness properties to hold in the $\mathcal{DY}$ intruder model, the assumption that the intruder does not disrupt (some of) the communication channels must therefore be added. A channel which guarantees to eventually deliver messages, even in the presence of the $\mathcal{DY}$ intruder, is called *resilient* (defined in § 5.4.2). Here we investigate the suitability of the $\mathcal{DY}$ intruder model for automatic verification of liveness properties with the resilient communication channels (RCC) assumption.

To impose RCC on the $\mathcal{DY}$ intruder, we recognise two possibilities: Either RCC can be applied as a fairness constraint such that the behaviours of the model which violate RCC are excluded from subsequent analyses, or we can change the specification of $\mathcal{DY}$ to an RCC-conforming intruder. First, we investigate the RCC assumption, and notice that some features of RCC are very complicated. In particular, resilient channels are sensitive to adding or removing prefixes, meaning that what has happened in the past determines what needs to be done in the future. For instance, if a message is sent to a resilient channel twice, then the message needs to be delivered twice. Whereas if the prefix of the execution which contains the first send action is removed from the system, then the message needs to be delivered

only once. This indicates that RCC does not directly fit in the standard notion of fairness constraints, i.e. those "insensitive to addition or deletion of prefixes" [Pra94]. Using RCC as a fairness constraint in verifying liveness properties would thus not be efficient, as complicated constraints often degrade the efficiency of verification techniques, cf. [EL86].

We present a modified intruder model, restricted by a much simpler fairness constraint, that is proved to be equivalent to a $\mathcal{DY}$ intruder that does not indefinitely delay the delivery of messages over resilient channels. In particular, for an arbitrary protocol and a liveness property, it is proved that if the $\mathcal{DY}$ intruder finds an attack (counterexample) for the property without violating the RCC assumption, then our proposed intruder model finds a corresponding attack, and vice versa. For safety properties, which do not require the RCC assumption, our intruder model is equivalent to $\mathcal{DY}$, after action renaming. Finally, we illustrate how liveness requirements of FE protocols can be checked using the proposed intruder model.

Liveness aspects of FE have often been left out from (mechanised) formal analyses, as in, e.g., [BP01, SM02, AB03, ES05a], see also our related work § 5.6. There are however at least two reasons why it is crucial to check liveness requirements of these protocols. Let us refer to them as the *theoretical reason* and the *pragmatic reason*. The theoretical reason is that FE inherently contains liveness properties. The non-termination flaws, in the protocols of, e.g., [ASW97, ZDB99, VPG01], respectively reported in [ASW98b, GRV05, Vog03], would not have been found if liveness of participants had been taken for granted. In fact, "identifying [termination] problem and providing an effective solution" to it is considered as "probably the most important contribution" of [ASW98b].

The pragmatic reason is that certain behaviours might be missing in protocol models or represented differently from their realisations. Any protocol in practice, arguably, only has finite runs (e.g. there are always time-out operations), and safety properties (stating that undesirable states are not reachable) may therefore seem sufficient. However, in modelling security protocols, timing aspects are usually abstracted away (see [Mea03] for a survey). For example, if a loop is mistakenly specified in Apache's rewrite module [Eng97], theoretically there can be a *livelock* in the server model that should be detected, even if the bogus server is not in practice endlessly trapped. As another example, a typical requirement for a FE protocol between Alice and Bob states that, if Alice receives a desired item from Bob, then Bob will eventually receive his desired item from Alice as well (see § 2.2). This is a liveness property that is violated if there is either a deadlock situation, such that Bob cannot progress after Alice has got her item, or a livelock situation, such that Bob runs into an endless execution (maliciously devised to prevent him to ever reach his goal). The latter situation sets up a threat alarm, although the realisation does not necessarily exhibit endless executions, e.g. Bob at some point gives up the exchange in an unfair state.

Road map § 5.2 contains definitions and notations that are used later in the chapter. In § 5.3 we present a formalisation of the RCC assumption. In § 5.5 we propose an intruder model that is proved to be equivalent to a $\mathcal{DY}$ intruder that respects the RCC assumption. As an application of the proposed intruder, formal verification of FE protocols is discussed in § 5.5.3

(verification case studies are reported in § 6). In § 5.6, we discuss our related work, and in § 5.7 we conclude the chapter.

5.2 Modelling security protocols

This section describes how we model protocols and intruders. We consider an asynchronous communication model with no global clock. A security protocol is modelled as an asynchronous composition of a finite number of processes with names. These processes model the roles of participants of the protocol. In a protocol, the set of process names is denoted $\mathcal{P}$. We overload $\mathcal{P}$ to also represent the protocol itself.

Processes communicate by sending and receiving messages. We let Msg be the set of all message contents that can be communicated (for a recursive formalisation of Msg see § 4.1.2). We assume that messages sent to the network are always tagged with the identity of the intended receiver of the message: $\langle q, m\rangle$, with $q \in \mathcal{P}$, denotes that message $m \in Msg$ is intended to be delivered to q. To send and receive a message $m \in Msg$ over the public communication channels, a process $p \in \mathcal{P}$ performs the actions $send_p(\langle q, m\rangle)$ and $recv_p(\langle p, m\rangle)$, respectively. [1] The public communication network net is also seen as a process (equipped with some internal buffer), that is external to $\mathcal{P}$.

Even though process p sends message m with the intention that it should be received by process q, it is in fact the network (the process net) that receives the message from p, and it is from the network that q can receive m. The communications between protocol participants and net are assumed to be synchronised (see § 4.1), meaning that p can receive a message m from net iff at the same time net can send m to p, and vice versa. It is assumed that net is always ready to receive messages from other processes. The communication between participants of a protocol via net is nevertheless asynchronous and a participant has in general no guarantees about the origins of the messages it receives.

Apart from $send$ and $recv$ actions [2], all other actions of processes in $\mathcal{P}$ are assumed not to communicate with any process outside $\mathcal{P}$. These can denote, e.g., internal decisions or communications between protocol participants through secure private channels (cf. § 4.1.2). To avoid name clashes, internal actions of different processes are assumed to be disjoint.

To model the $\mathcal{DY}$ intruder that has complete control over the network, we assume that it plays the role of the network (i.e. net). The intruder may thus schedule messages and possibly insert its own messages into the network. Besides *being* the network, the intruder can also have legitimate roles in protocols (which is required, e.g., in modelling FE protocols). The

[1] The process name p is redundantly used in $recv_p(\langle p, m\rangle)$ only to ensure the semantic consistency between the data terms of $send$ and $recv$ actions.

[2] The subscripts of $send_p$ and $recv_p$ actions are suppressed when the context is clear, or when the discussion holds for any $p \in \mathcal{P}$.

following process specifies $\mathcal{DY}$ (recall the notations introduced in § 4.1).

$$\begin{aligned}\mathcal{DY}(\Gamma : Set) = \quad & \Sigma_{q\in\mathcal{P},m\in Msg} \\ & \qquad recv_I(\langle q, m\rangle) \cdot \mathcal{DY}(\{m\} \bigcup \Gamma) \\ & + \\ & \Sigma_{q\in\mathcal{P},m\in Msg} \\ & \qquad send_I(\langle q, m\rangle) \cdot \mathcal{DY}(\Gamma) \lhd synth(m, \Gamma) \rhd \delta\end{aligned}$$

As we focus on the concurrency aspects of the $\mathcal{DY}$ process, its data manipulation capabilities, namely $\bigcup$ and $synth$ functions, are left unspecified here (a formal specification of these is presented in § 4.1.2). They are however assumed to implement the deduction rules of definition 2.2. The set Γ represents $\mathcal{DY}$'s knowledge. We write Γ_0 for the initial value of Γ (usually containing process identities, public keys, private keys of corrupted parties, etc.)

To facilitate communications, for every process $p \in \mathcal{P}$ we define

$$\begin{aligned} recv_I | send_p &= \mathbf{send}_p \\ send_I | recv_p &= \mathbf{recv}_p \end{aligned}$$

A merit of using synchronous communication between $\mathcal{P}$ and $\mathcal{DY}$ (as it plays the role of net) is that $\mathcal{DY}$ cannot send messages which $\mathcal{P}$ does not accept, simply because synchronisations with the corresponding actions of $\mathcal{P}$ would fail (cf. § 4.1.2). This allows us to inductively define the (infinite) set Msg of possible messages in a generic way, so that the intruder specification does not depend on the protocol being analysed.

For an example of a protocol specification in the presence of $\mathcal{DY}$ we refer to § 4.1.2. In the rest of this chapter, we study LTSs L that result from the interactions between the $\mathcal{DY}$ intruder and a protocol $\mathcal{P}$, i.e. $L = \partial_H(P \| \mathcal{DY}(\Gamma_0))$, where $H = \{send_p, recv_p \mid p \in \mathcal{P}\} \cup \{send_I, recv_I\}$, P denotes the initial state of $\mathcal{P}$ and Γ_0 is the initial knowledge set of the intruder. No assumptions are put on acyclicity or determinacy of L. We however assume that L is finite. Besides, since in single-image LTSs a trace corresponds to a unique sequence of states, we focus on single-image LTSs in the following to simplify the presentation. Our results (e.g. theorem 5.1) do also hold in LTSs which are not single-image. This is intuitively because we present translations between LTSs which map a single sequence of states to another one, in single-image LTSs. These translations would instead simply map sets of sequences of states to each other in case LTSs were not single-image.

5.3 Safety, liveness and the $\mathcal{DY}$ intruder

Requirements of systems can usually be divided into *safety* and *liveness* classes. Below, we define these classes and investigate how these requirements are achievable for security protocols in the presence of $\mathcal{DY}$.

Let A be a countable set. A^* and A^ω denote the set of finite and infinite sequences over A, respectively. Below, the notion of traces is used to refer to *any* sequence of elements in A,

cf. § 4.1.1. Let A^∞ be the set of all traces of the elements of A, namely $A^\infty = A^* \cup A^\omega$. The empty trace is denoted by ϵ. A *trace property* ϕ is a recursive subset of A^∞, thus being characterised by a function $\phi : A^\infty \to Bool$. The property $\neg\phi$ is defined as $A^\infty \setminus \phi$. Trace properties can be divided into two classes: Safety properties (stating that something bad will never happen) and liveness properties (stating that something good will eventually happen). Below, we formally define these classes. Our definitions are slightly different from [AS84], because we use LTSs to describe our models, while total Kripke structures are used in [AS84].

For a trace $\alpha = \alpha_1 \cdot \alpha_2 \cdots \alpha_i \cdots$ we write α^i for the trace $\alpha_1 \cdot \alpha_2 \cdots \alpha_i$, when i is less than or equal to the length of α. The concatenation of α and α' is denoted $\alpha \cdot \alpha'$.

5.1. DEFINITION. *A trace property ϕ is a safety property iff*

$$\forall \alpha \in A^\infty.\, \neg\phi(\alpha) \implies (\exists i. \forall \alpha' \in A^\infty.\, \neg\phi(\alpha^i \cdot \alpha')).$$

5.2. DEFINITION. *A trace property ϕ is a liveness property iff*

$$\forall \alpha \in A^*. \exists \alpha' \in A^\infty.\, \phi(\alpha \cdot \alpha').$$

It is worth mentioning that safety properties are closed sets in the natural topology on A^ω, while liveness properties are dense sets [AS84]. Therefore, any trace property can be expressed as the intersection of a safety property and a liveness property [AS84].

Any LTS defines a trace property by itself. Given an LTS $L = (S, s_0, A, Tr)$, [3] the trace property defined by L is $\pi(s_0)$. For example, the LTS corresponding to process $p = a \cdot b \cdot \delta$, with $a, b \in A$, defines the trace property $\{\epsilon, a, a \cdot b\}$. For a trace property ϕ and state s, we write $s \models \phi$ iff $\pi(s) \subseteq \phi$. When $s_0 \models \phi$, we may write $L \models \phi$. In other words, L satisfies ϕ iff the property defined by L is contained in ϕ. Note that if ϵ was not a member of each $\pi(s)$, then deadlock states would undesirably satisfy any (liveness) trace property. [4]

We remark that complements of *some* liveness properties turn out to be safety properties and vice versa [Pra94]. This however does *not* imply that checking, say, liveness properties can be reduced to checking safety. This is because, although $L \models \neg\phi$ implies $L \not\models \phi$, indeed $L \not\models \neg\phi$ does not imply $L \models \phi$, hence being in general inconclusive.

A *fairness constraint* is a function which given an LTS L maps traces of $\pi(s_0)$ to $\{\mathsf{T}, \mathsf{F}\}$, cf. the $\mathcal{F}_0$ constraint of definition 4.1. Note that there is a slight difference between fairness constraints and trace properties: Fairness constraints accept (or reject) a trace, in general, depending on the LTS in which the trace occurs, while trace properties are defined independent of any particular LTS, cf. [VV06]. To define fairness constraints independent of LTSs, we could annotate traces with their corresponding ready transitions (see § 4.1.1). [5] For a trace property ϕ and fairness constraint $\mathcal{F}$, we write $s \models_{\mathcal{F}} \phi$ iff $\forall \alpha \in \pi(s).\, \mathcal{F}(\alpha) \implies \phi(\alpha)$.

[3] In general, A in L can be different from the set based on which the properties are defined. We can however without loss of generality consider their union in both definitions.

[4] We can think of ϵ as a technical means to mimic the deadlock state stuttering of [AS84].

[5] Fairness constraints in general need not be expressed in terms of trace properties. These are defined, for instance, using CTL formulae in the SMV model checker [McM00, CGP00]. The annotation method mentioned above does not cover fairness in such cases, as CTL is a branching logic.

In verifying liveness properties, note that only maximal traces of L are of interest. This is because intuitively liveness talks about a "good" thing which eventually occurs. Therefore, a finite trace that can still be extended in L does not constitute an evidence that the "good" thing is not forthcoming. This maximality condition can be seen as a fairness constraint. We say a trace α in L is $\mathcal{F}_1$-fair iff it is maximal in L. Note that $\mathcal{F}_0(\alpha) \implies \mathcal{F}_1(\alpha)$, for any LTS L and α in L (see § 4.1.1 for the definition of $\mathcal{F}_0$).

Most security properties can be expressed as safety properties. Secrecy, for instance, is a trace property only consisting of traces which do not contain action $\mathrm{secret.revealed}$, with $\mathrm{secret.revealed}$ denoting the point a secret is revealed to the intruder. There are however security properties which cannot be encoded as safety properties. For instance consider the following goal auth for authentication protocols: If A terminates a run of protocol $\mathcal{P}$ apparently with B, then B will terminate the same run of protocol $\mathcal{P}$ apparently with A. It is easy to see that authentication as defined above is a liveness property: Any trace $\alpha \in A^*$ that does not belong to auth can be extended with $\mathrm{B.authenticates.A}$ action to satisfy auth. [6]

As another example, let action $\mathrm{available}$ denote the point where a web service is ready to serve clients. A liveness property containing all the traces in which $\mathrm{available}$ occurs infinitely often is the most natural candidate to formalise resisting denial of service attacks. Similarly, in FE, the termination requirement can be encoded as a liveness, but not as a safety, property.

The $\mathcal{DY}$ intruder model has originally been devised to check safety properties of security protocols. Liveness properties in general do not hold in this model, as $\mathcal{DY}$ can deliberately disrupt all communication channels. To check liveness properties in the $\mathcal{DY}$ model, some modifications to the model are thus necessary.

In practice, to uphold liveness properties, resilient channels are usually assumed. These channels guarantee to eventually deliver messages transmitted over them intact (cf. § 2.2.2 and § 3.2). We refer to this assumption as RCC. Back to the $\mathcal{DY}$ model, RCC implies that $\mathcal{DY}$ cannot destroy messages transmitted over resilient channels. $\mathcal{DY}$ can however delay and shuffle these, inject fabricated messages in between them, and so forth. We recognise two possibilities to embed RCC in the $\mathcal{DY}$ model: We can either impose RCC as a fairness constraint such that the behaviours of the model which violate RCC are excluded from subsequent analyses, or we can change the specification of the $\mathcal{DY}$ process to an RCC-conforming intruder. In the following, we investigate these possibilities.

5.4 Resilient communication channels assumption

> People never give your message to anybody.
>
> J. D. Salinger's *The catcher in the rye* (1951)

[6] Authentication is usually specified as a safety property in the literature, stating that when A terminates a run of protocol $\mathcal{P}$ apparently with B, then B should have been in some way involved in this run of $\mathcal{P}$, cf. [Low97]. For an authentication protocol which fails to achieve auth, as defined above, see [Syv94].

This section discusses resilient channels from a practical point of view, to provide some insight and motivations for our modelling choices.

In heterogeneous networks, such as the Internet, communication channels can be faulty. They may for instance lose, duplicate and reorder messages transmitted over them. Intuitively, no non-trivial liveness property of protocols can be guaranteed with faulty channels. This is because messages sent over these channels may all be discarded. The protocol should therefore achieve its goals with no public communications. Such goals amount to trivial properties in asynchronous systems.

In order to guarantee liveness, stronger assumptions on communication channels are required. For instance, most optimistic FE protocols rely on resilient channels, see § 2.2.2. According to Asokan "a message inserted into a resilient channel will eventually be delivered" [Aso98]. Although this is an asymptotic restriction, i.e. no bounds are placed on the order or the time of delivering messages, resilient channels are not available in most practical situations. Available faulty channels can nonetheless be used to provide resilience, as described below. Assuming RCC in security protocol thus helps us to abstract from the underlying mechanisms which actually provide resilience.

There are various ways to construct resilient channels from faulty ones. Let us assume that A and B are connected with a faulty channel c which may lose, duplicate and reorder messages. To distinguish c from a transient channel, we assume that there is a bound on the number of messages that c can discard. We say that a channel is *fair lossy* iff any message which is inserted to one end of the channel an infinite number of times, is delivered to the other end of the channel an infinite number of times. [7] If c is a fair lossy channel, which may duplicate and reorder messages, then, essentially, retransmission and tagging allow A and B to construct a reliable FIFO channel on top of c, e.g. see Stenning's protocol [Ste76, Lyn96].

In the $\mathcal{DY}$ intruder model, it is assumed that the only possible means of communication between A and B is $\mathcal{DY}$. The $\mathcal{DY}$ intruder, however, need not be a fair lossy medium and can destroy all the messages that are transmitted through it. Therefore, no reliable channel may be constructed between A and B in the $\mathcal{DY}$ model. Nevertheless, the assumption that $\mathcal{DY}$ controls *all* communication media between A and B is often impractical. For instance, in wireless networks, given that jamming is only locally sustainable, A and B can always move to an area where they can send and receive messages. Ultimately, two principals who fail to properly establish a channel over computer networks can resort to other communication means, such as various postal services. These services, albeit being orders of magnitude slower than computer networks, are very reliable and well protected by law.

We thus postulate that any A and B who are willing to communicate can eventually establish a (fair lossy) channel, despite $\mathcal{DY}$'s obstructions. To add this postulation to the $\mathcal{DY}$ model, weakening $\mathcal{DY}$ to the extent that it behaves as a fair lossy channel is adequate.

As mentioned earlier, many security protocols abstract away from the underlying mech-

[7] This corresponds to the *strong loss limitation* condition in [Lyn96]. A weaker variant of this requirement states that if an infinite number of messages are sent to the channel, then *some* infinite subset of them are delivered.

anism of providing reliable channels and, instead, rely on RCC. Although a fair lossy $\mathcal{DY}$ might provide a suitable level of abstraction for checking, e.g., Stenning's protocol, for checking higher level security protocols we need a coarser abstraction for an RCC-conforming $\mathcal{DY}$. This is the topic of the next section.

5.4.1 Embedding RCC in the $\mathcal{DY}$ model: An intuitive description

To structure our intuitive understanding of RCC, we make the following choices explicit:

R1. $\mathcal{DY}$ not only eventually delivers all messages transmitted over resilient channels, but it is also infinitely often ready to accept messages being sent to resilient channels.

R2. A message sent to a resilient channel may be "discarded" (by $\mathcal{DY}$) only when it has been delivered to its recipient or is from some time onwards never accepted by its purported recipient.

Since we have assumed that the role of the communication network is played by a process, namely $\mathcal{DY}$, we need to ensure that this process does receive messages on resilient channels. R1 reflects this modelling decision. In practical terms, R1 may be read as: Resilient channels cannot be removed from the network. They may however be temporarily unavailable.

R2, although innocent looking, can cause counter-intuitive behaviours in case protocols directly exploit the buffer resilient channels provide. For instance, consider a protocol involving two processes p and q specified as $p = send_p(\langle q, m_1\rangle) \cdot send_p(\langle q, m_2\rangle) \cdot \delta$ and $q = recv_q(\langle q, m_2\rangle) \cdot recv_q(\langle q, m_1\rangle) \cdot \bar{\imath} \cdot \delta$. Note that action $\bar{\imath}$ is reached only if the communication channel connecting p to q has a buffer, so that the messages can be delivered in the reverse order. This may seem to be a spurious behaviour in the system. We contend that in a real network, messages can be shuffled such that $\bar{\imath}$ is reached. This shuffling is also allowed in our model, because of R2, considering it as a consequence of asynchronous communication and network interface buffers.

Substituting R2 with a condition which requires delivering messages only in the right order would enable channels to discard more messages and still be considered resilient. This modelling alternative would thus result in a weaker condition compared to the resilient channels à la Asokan [Aso98]. Consequently, strictly stronger intruder models can be obtained, the practical relevance of which might be interesting to investigate. A theoretical question pertinent to this observation is determining the weakest resiliency condition that is needed to achieve fair exchange, cf. [GHK+07].

We note that R2 does not prevent the channel (played by $\mathcal{DY}$) to wait till a message becomes undeliverable and then destroy it. For instance, when process $q = recv_q(\langle q, m_1\rangle).\delta + recv_q(\langle q, m_2\rangle).\delta$ is waiting to receive a message and messages m_1 and m_2 are both buffered in the channel, after sending m_1 to q, message m_2 can be destroyed without violating R2.

Assuming R2 (or, instead, assuming that resilient channels would never destroy messages) implies that no bounds can a priori be placed on a channel's buffer size. For instance,

consider a protocol consisting of two processes specified as $p(n : \mathbb{N}) = send_p(\langle q, n\rangle) \cdot p(n-1) \lhd n > 0 \rhd \delta$, and $q(n, n' : \mathbb{N}) = recv_q(\langle q, n\rangle) \cdot q(n+1, n') \lhd n < n' \rhd recv_q(\langle q, n'\rangle) \cdot \bar{\imath} \cdot \delta$. For any $n' \in \mathbb{N}$, processes $q(1, n')$ an $p(n')$, communicating over a resilient channel, can produce action $\bar{\imath}$ according to R2, while the channel's buffer size is n', hence unbounded. This is a well-known fact that implementing reliable channels in asynchronous systems requires infinite storage for buffering messages. Finite buffers may overflow and thus cause message losses, cf. [BCT96]. No channel with an infinite buffer can practically be realised. We thus associate a loose buffer size to RCC.

R3. For a given $n \in \mathbb{N}$, a channel that realises RCC_n (called an RCC_n channel) guarantees that, if a message is sent n' times to the channel, then the channel will eventually deliver at least $min(n, n')$ instances of the message intact to its destination (if the message has a recipient).

Note that an RCC_n channel's buffer may in general not be finite, if the number of distinct messages submitted to the channel are infinite.

RCC_n is clearly a weaker constraint than RCC. In fact, for each n, we can construct a protocol such that when assuming RCC_n action $\bar{\imath}$ is not produced in the protocol, while with RCC it is. For instance, consider the aforementioned processes $p(n')$ and $q(1, n')$, which cannot produce $\bar{\imath}$ while assuming RCC_n, with $n < n'$. These can however produce $\bar{\imath}$, when assuming RCC. Therefore, if a protocol achieves a liveness goal in an RCC_n network, it will also achieve it in any RCC_m network for $m \geq n$. To simplify our formalisation, from this point on, we focus on RCC_1. Confining to RCC_1 also comes to terms with practice as it is not advised to use the same message for different purposes in a protocol [AN96].

5.4.2 RCC as a fairness constraint

First we recapitulate our reasoning steps: Let $\mathcal{P}$ be a protocol aiming at a liveness property ϕ, say, the inevitable reachability of $\bar{\imath}$. The LTS describing $\partial_H(P \| \mathcal{DY}(\Gamma_0))$ is called L (see § 5.2). An attack on $\mathcal{P}$ is a trace α in L such that $\alpha \notin \phi$, e.g. a trace belonging to L which does not contain $\bar{\imath}$. We observe that ϵ belongs to any L, while it belongs to no ϕ (assuming that ϕ is a non-trivial liveness property). Therefore, certain fairness constraints need to be applied such that only legitimate traces of L are compared against ϕ.

To allow maximal progress for honest participants we would for instance require that any legitimate trace needs to be maximal, namely fair à la $\mathcal{F}_1$ (see § 5.3 for the definition of $\mathcal{F}_1$), cf. [ACC07]. Otherwise, an honest Bob who does not receive a certain message from Alice in the optimistic protocol might never try contacting the TTP (see § 3). Fairness à la $\mathcal{F}_0$ seems also to be a natural requirement for legitimate traces. This is to exclude any trace in which a process is never scheduled. Below, for a trace α in L, we define a fairness constraint that determines whether $\mathcal{DY}$ respects RCC_1 in α or not. We will see that $\mathcal{F}_0$ or $\mathcal{F}_1$, although relevant, are not the proper constraints.

Motivations

We first motivate why simply $\mathcal{F}_0$ or $\mathcal{F}_1$ cannot be used as RCC_1. This discussion also shows subtleties in precisely capturing RCC_1.

Observe that the $\mathcal{DY}$ process as defined in § 5.2 is not allowed to remove messages from Γ. Therefore, restricting L's behaviours to only fair traces à la $\mathcal{F}_1$ does not properly implement RCC_1. The following example clarifies the problem. Below, let $\pi^\infty(s)$ be the set of maximal traces in $\pi(s)$.

5.1. EXAMPLE. *Let $p = send_p(\langle q, t_1\rangle) \cdot \delta$ and $q = recv_q(\langle q, h(t_1)\rangle) \cdot \bar{\imath} \cdot \delta$. Consider the system $L = \partial_H(p \| q \| \mathcal{DY}(\emptyset))$. Let ϕ be the set of traces that contain $\bar{\imath}$. We derive $\pi^\infty(s_0) = \{\mathbf{send}_p(\langle q, t_1\rangle).\mathbf{recv}_q(\langle q, h(t_1)\rangle).\bar{\imath}\}$. Indeed, $s_0 \models_{\mathcal{F}_1} \phi$, hence $L \models_{\mathcal{F}_1} \phi$. This result is absurd, because the protocol described by p and q is defunct. The action $\bar{\imath}$ is reached only because the fairness constraint that we impose, namely the maximality constraint $\mathcal{F}_1$, forces $\mathcal{DY}$ to, in some sense, help the participants.* [8]

A similar behaviour can be observed when considering $\mathcal{F}_0$ in protocols with cyclic processes. This is demonstrated in the following example.

5.2. EXAMPLE. *Let $p = send_p(\langle q, t_1\rangle) \cdot p$ and $q = recv_q(\langle q, t_1\rangle) \cdot q + recv_q(\langle q,\ t_1, t_1\rangle) \cdot \bar{\imath} \cdot \delta$ in the setting of example 5.1. We note that $\mathbf{send}_p^\omega(\langle q, t_1\rangle) \in \pi^\infty(s_0)$. Therefore, $\pi^\infty(s_0) \not\subseteq \phi$, implying $s_0 \not\models_{\mathcal{F}_1} \phi$. This trace is however not fair according to $\mathcal{F}_0$, simply because process q is never scheduled. In fact the only $\mathcal{F}_0$-trace in $\pi(s_0)$ is the following trace:*

$$\mathbf{send}_p(\langle q, t_1\rangle).(\mathbf{send}_p(\langle q, t_1\rangle) \vee \mathbf{recv}_q(\langle q, t_1\rangle))^*.\mathbf{recv}_q(\langle q,\ t_1, t_1\rangle).\\ \mathbf{send}_p^*(\langle q, t_1\rangle).\bar{\imath}.\mathbf{send}_p^\omega(\langle q, t_1\rangle)$$

Therefore, $s_0 \models_{\mathcal{F}_0} \phi$. This is absurd, as reaching $\bar{\imath}$ in the protocol is a courtesy of $\mathcal{DY}$.

Apparently, if the $\mathcal{DY}$ process could explicitly destroy parts of its knowledge Γ, then it could destroy t_1 in example 5.1 to avoid being forced by $\mathcal{F}_1$ to send $h(t_1)$ to q. This would however not solve the problem of example 5.2, because process p constantly fills Γ. The following example explores this direction.

5.3. EXAMPLE. *In the setting of example 5.2, we define an intruder process which can ex-*

[8] "Forcing" the intruder only arises in modelling by ignoring other possible behaviours of the intruder.

plicitly remove elements of its knowledge (cf. § 5.2):

$$\begin{aligned}\mathcal{DY}_r(\Gamma : Set) = \quad & \Sigma_{q\in\mathcal{P},m\in Msg} \\ & \qquad recv_I(\langle q,m\rangle)\cdot\mathcal{DY}_r(\{m\}\bigcup\Gamma) \\ & + \\ & \Sigma_{q\in\mathcal{P},m\in Msg} \\ & \qquad send_I(\langle q,m\rangle)\cdot\mathcal{DY}_r(\Gamma)\lhd synth(m,\Gamma)\rhd\delta \\ & + \\ & \Sigma_{m\in\Gamma} \\ & \qquad rem_I(m)\cdot\mathcal{DY}_r(\Gamma\setminus\{m\})\end{aligned}$$

Figure 5.1 depicts the LTS corresponding to $L = \partial_H(p\|q\|\mathcal{DY}_r(\emptyset))$. Observe that the $\mathcal{F}_0$-fair traces in $\pi(s_0)$ indeed contain $\bar{\imath}$.

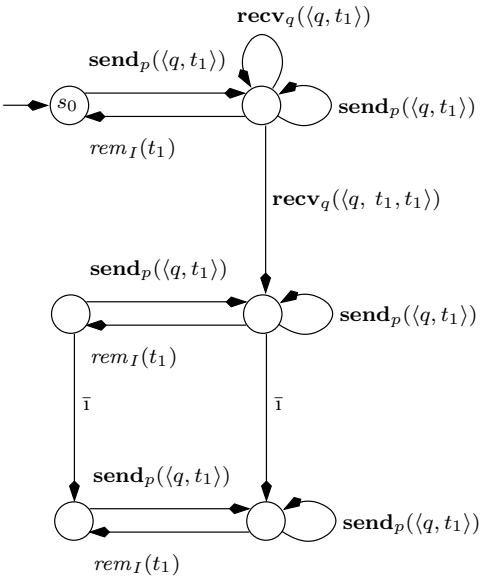


Figure 5.1: *LTS of the protocol of example 5.3.*

Formalisation

The previous examples suggest that RCC_1 is similar to $\mathcal{F}_0$ and $\mathcal{F}_1$. However, $\mathcal{F}_0$ and $\mathcal{F}_1$ are too *strong* to be included in RCC_1. Namely, according to $\mathcal{F}_0$ any possibility which is infinitely often available, should be infinitely often realised. In contrast, RCC_1 does allow indefinitely ignoring a possibility which is infinitely often available, if the possibility is, e.g.,

delivering a message which has never been submitted to the network (the pair t_1, t_1 in example 5.2). Similarly, $\mathcal{F}_1$ considers a finite trace fair only if it is maximal, while this condition may force $\mathcal{DY}$ to deliver a message beyond what RCC_1 would require (e.g. delivering message $h(t_1)$ in example 5.1). As we will show, the difficulty in having RCC as a fairness constraint lies mainly in a need to look back and check the actions that have happened.

Below we characterise the set of actions that an RCC_1 channel can *ignore*. This set contains the set of actions that deliver messages which have not been sent to the channel after their last delivery (if ever).

5.3. DEFINITION. *Associated to trace $\alpha = \alpha_1 \cdot \alpha_2 \cdots$ in LTS $L = (S, s_0, A, Tr)$, we define the sequence $\Theta^\alpha = \Theta_0^\alpha \cdot \Theta_1^\alpha \cdots$ as the following: (recall from § 4.1.1 that $s^\alpha = s_0^\alpha \cdot s_1^\alpha \cdots$)*

$$\Theta_i^\alpha = \{(s_i^\alpha, \mathbf{recv}_q(\langle q, m\rangle), s') \in Tr \mid \forall p \in \mathcal{P}, j < i.\ \alpha_j = \mathbf{send}_p(\langle q, m\rangle) \implies \exists j < k < i.\ \alpha_k = \mathbf{recv}_q(\langle q, m\rangle)\}$$

Note that $\Theta_i^\alpha \subseteq T_i^\alpha$, for any i. Now, we are ready to define RCC_1 as a fairness constraint.

5.4. DEFINITION. *Trace $\alpha = \alpha_1 \cdot \alpha_2 \cdots$ in LTS $L = (S, s_0, A, Tr)$ is fair according to RCC_1, called $\mathcal{F}_r$-fair, iff*

- *If α is finite, either it is maximal or it ends in a state where only transitions from Θ^α are available, namely:*

$$\alpha = \alpha_1 \cdots \alpha_n \implies T_n^\alpha \setminus \Theta_n^\alpha = \emptyset$$

- *If α is infinite, then for each $\theta \in Tr$, if $\{i \mid \theta \in T_i^\alpha \setminus \Theta_i^\alpha\}$ is infinite, then so is $\{i \mid \theta = t_i^\alpha\}$.*

We note that, in general, $\mathcal{F}_r(\alpha)$ does not imply $\mathcal{F}_0(\alpha)$, simply because $T_i^\alpha \setminus \Theta_i^\alpha \subseteq T_i^\alpha$. Similarly, $\mathcal{F}_r(\alpha)$ does not imply $\mathcal{F}_1(\alpha)$, because $\emptyset \subseteq \Theta_i^\alpha$. Below, we see how conditions R1, R2 and R3 are met when applying $\mathcal{F}_r$.

- In the $\mathcal{DY}$ process, $recv_I$ action is always available. Moreover, **send** actions never appear in Θ, hence always being treated fairly. R1 is thus satisfied in $\mathcal{F}_r$-fair traces.
- While a submitted message is not delivered, **recv** for that message does not appear in Θ. If the message has a recipient, then **recv** corresponding to the message appears in T, hence $\mathcal{F}_r$ forcing $\mathcal{DY}$ to eventually deliver the message. Now, R3 states that if the message is delivered once, $\mathcal{DY}$ does not need to deliver it again unless it is submitted to the channel again. In fact, this exact condition is encoded in Θ. Therefore, $\mathcal{F}_r$ satisfies R2 and R3.

RCC_1, as specified above, is a complicated condition. In particular, computing Θ requires looking back deep in the past and, moreover, Θ depends on message contents, which are not a priori known when analysing security protocols. The following example shows how finite state transducers can be used to compute Θ.

5.4. EXAMPLE. *Given a trace α in LTS L, we define a function to characterise Θ^α, i.e. to determine at each state s_i whether a certain* **recv** *belongs to Θ_i or not. For each message m that appears as a parameter of actions in α, let $a_{m,q} = \mathbf{recv}_q(\langle q, m\rangle)$ and $\bar{a}_{m,q} = \mathbf{send}_p(\langle q, m\rangle)$, for any $p \in \mathcal{P}$. For each such $a_{q,m}$ we construct a finite state transducer as depicted in figure 5.2. This transducer, given α as input, produces a sequence of* T *and* F *values with which we label the states of s^α.*

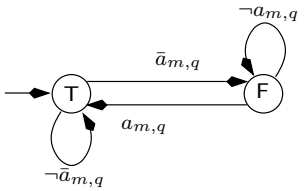


Figure 5.2: *Finite state transducer of example 5.4.*

To determine whether at state s_i, transition $(s_i, a_{m,q}, s')$ belongs to Θ_i or not, we look at the label that the corresponding transducer would assign to s_i: A T *means that the transition belongs to Θ_i (and conversely for* F*). In LTS $(\{s_0, s_1, s_2, s_3\}, s_0, \{a, \bar{a}\}, Tr)$ with $Tr = \{(s_0, \bar{a}, s_1), (s_1, a, s_2), (s_2, a, s_3)\}$, for trace $\alpha = \bar{a} \cdot a \cdot a$ we have* T, F, T, T *assigned respectively to s_0, s_1, s_2, s_3. Therefore, RCC_1 does not require taking action a at state s_2, as the corresponding transition belongs to Θ_2. Note that to model* RCC *(with unbounded buffer) a push-down transducer would be required.* [9]

Intuitively, $\mathcal{F}_r$ is not local, in the sense that to decide whether a certain transition should be treated fairly or not, we need to look far back in the trace, and a priori we do not know how far. Below, we formalise this intuition using the *local testability* notion of [MP71].

Given a finite trace $\alpha \in A^*$ and a natural number k, define $l_k(\alpha)$, $r_k(\alpha)$ and $i_k(\alpha)$ as, respectively, the left-end segment of α of length k, the right-end segment of α of length k, and the set of interior segments of α of length k. When the length of α is less than or equal k, then $l_k(\alpha) = r_k(\alpha) = \alpha$ and $i_k(\alpha) = \emptyset$. We say that a trace property ϕ is k-*testable* iff $\forall \alpha, \alpha' \in A^*$, if $l_k(\alpha) = l_k(\alpha')$, $r_k(\alpha) = r_k(\alpha')$ and $i_k(\alpha) = i_k(\alpha')$, then $\alpha \in \phi \iff \alpha' \in \phi$. We say ϕ is *locally testable* if it is k-testable for a natural $k > 0$.

Roughly speaking, monitoring a trace through a window of a bounded size and recording the observations (without counting or preserving any order) is enough to determine whether the trace belongs to a locally testable property or not. Locally testable properties are a proper subset of star-free regular languages and can be implemented using relatively simple loop-free circuits [MP71]. Star-free regular languages in turn are as expressive as LTL confined to finite traces (see, e.g., [Eme90]). Below, we show that $\mathcal{F}_r$ is not locally testable.

[9] Our use of finite state transducers should not be confused with the way finite state acceptors (usually Büchi automata) are used to encode temporal properties, as in, e.g., [VW86].

5.1. LEMMA. *$\mathcal{F}_r$ is not locally testable.*

Proof: Assume, towards a contradiction, that $\mathcal{F}_r$ is locally testable. Therefore, $\mathcal{F}_r$ is k-testable for a natural number k. Let $\{a, \bar{a}, \imath\} \subseteq A$ in the context of example 5.4. Consider the following two traces α and α' belonging to LTS L.

$$\alpha = \imath^k.\bar{a}.\imath^k.\bar{a}.\imath^k.a.\imath^k$$
$$\alpha' = \imath^k.\bar{a}.\imath^k.a.\imath^k.\bar{a}.\imath^k$$

Assume further that α and α' are not maximal in L. Note that $\alpha \in \mathcal{F}_r$ and $\alpha' \notin \mathcal{F}_r$ in L. Observe that $l_k(\alpha) = l_k(\alpha') = \imath^k$ and $r_k(\alpha) = r_k(\alpha') = \imath^k$. Similarly, $i_k(\alpha) = i_k(\alpha') = \{\imath^k\} \cup_{0 \leq j \leq k-1} \{\imath^{k-j-1} a\, \imath^j\} \cup_{0 \leq j \leq k-1} \{\imath^{k-j-1} \bar{a}\, \imath^j\}$. Therefore, according to k-testability of $\mathcal{F}_r$, we should have $\mathcal{F}_r(\alpha) \iff \mathcal{F}_r(\alpha')$. Thus a contradiction is reached. Intuitively, since instances of a and $\bar{a}$ are placed at least k points apart, an observer watching through a window of width k cannot detect that, e.g., in α no a occurs between the two $\bar{a}$. ■

In verification, simple fairness constraints are preferred for efficiency reasons, cf. [EL86]. In the following, we describe a modified $\mathcal{DY}$ process, so that a simpler fairness constraint can capture its RCC_1-conforming behaviours. The constraint proposed in the next section is locally testable and it does not depend on message contents.

5.5 An intruder model for verifying liveness

We consider an arbitrary protocol $\mathcal{P}$ and a liveness property ϕ. Our goal is to verify whether $\partial_H(P \| \mathcal{DY}(\Gamma_0)) \models_{\mathcal{F}_r} \phi$ holds or not. In this section, we propose an intruder model $I^\dagger$, such that

$$\partial_H(P \| \mathcal{DY}(\Gamma_0)) \models_{\mathcal{F}_r} \phi \iff \partial_H(P \| I^\dagger(\Gamma_0, \emptyset)) \models_{\mathcal{F}^\dagger} \phi,$$

with fairness constraint $\mathcal{F}^\dagger$ being much simpler than $\mathcal{F}_r$.

The intruder model $I^\dagger$ is presented in § 5.5.1 and we study the relation between the LTSs resulting from $\partial_H(P \| \mathcal{DY}(\Gamma_0))$ and $\partial_H(P \| I^\dagger(\Gamma_0, \emptyset))$ in § 5.5.2. Encoding $\mathcal{F}^\dagger$ in alternation-free regular μ-calculus is the topic of § 5.5.3.

5.5.1 Modified intruder model

The modified intruder process $I^\dagger$ is the same as $\mathcal{DY}$ (see § 5.2), except that besides Γ, we use another set $\mathcal{B}$ to store the messages that have been received but not yet sent by the intruder. [10] Moreover, a separate send action, $send_I^\dagger$, for sending messages not belonging to $\mathcal{B}$, is used to

[10] For providing RCC_n, with $n > 1$, $\mathcal{B}$ can be modelled as a multi-set.

detect and avoid behaviours of the intruder that are not required by RCC_1.

$$
\begin{aligned}
I^\dagger(\Gamma, \mathcal{B} : Set) = \quad & \Sigma_{q\in\mathcal{P}, m\in Msg} \\
& \qquad recv_I(\langle q, m\rangle) \cdot I^\dagger(\{m\} \bigcup \Gamma, \{\langle q, m\rangle\} \cup \mathcal{B}) \\
& + \\
& \Sigma_{q\in\mathcal{P}, m\in Msg} \\
& \qquad send_I(\langle q, m\rangle) \cdot I^\dagger(\Gamma, \mathcal{B} \setminus \{\langle q, m\rangle\}) \lhd \langle q, m\rangle \in \mathcal{B} \rhd \delta \\
& + \\
& \Sigma_{q\in\mathcal{P}, m\in Msg} \\
& \qquad send^\dagger_I(\langle q, m\rangle) \cdot I^\dagger(\Gamma, \mathcal{B}) \lhd synth(m, \Gamma) \wedge \langle q, m\rangle \notin \mathcal{B} \rhd \delta
\end{aligned}
$$

We let the initial state of $I^\dagger$ be $(\Gamma_0, \emptyset)$, and for all $p \in \mathcal{P}$, we let $send^\dagger_I | recv_p = \mathbf{recv}^\dagger_p$. Moreover, the set H is extended to also include $send^\dagger_I$ (cf. § 5.2).

As earlier, a fairness constraint is needed to restrict the behaviour of the intruder. Intuitively, set $\mathcal{B}$ plays the role of the resilient channels' buffer, and action $\mathbf{recv}^\dagger$ marks the elements of Θ. RCC for $I^\dagger$ can thus be stated with no reference to Θ or message contents. Below, as a convention, when $t = s \xrightarrow{a} s'$, we use $\lambda(t)$ and $\lambda(a)$ interchangeably, and likewise for sets of transitions and sets of actions (see the definition of λ in § 4.1).

5.5. DEFINITION. *Trace $\alpha = \alpha_1 \cdot \alpha_2 \cdots$ in LTS $L = (S, s_0, A, Tr)$ is $\mathcal{F}^\dagger$-fair iff*

- *If α is finite, either it is maximal or it ends in a state where only $\mathbf{recv}^\dagger$ actions are available, namely:*

$$\alpha = \alpha_1 \cdots \alpha_n \implies \lambda(T^\alpha_n) \subseteq \{\mathbf{recv}^\dagger\}$$

- *If α is infinite, then for each θ such that $\lambda(\theta) \neq \mathbf{recv}^\dagger$, if $\{i \mid \theta \in T^\alpha_i\}$ is infinite, then so is $\{i \mid \theta = t^\alpha_i\}$.*

This fairness constraint can informally be stated as: No possibilities enabled infinitely often, except for $\mathbf{recv}^\dagger$ actions, may be excluded forever. Note that $\mathcal{F}^\dagger$ does not depend on message contents. Moreover, in the definition of $\mathcal{F}^\dagger$, we can locally determine at each state whether an outgoing transition needs to be treated fairly or not. This is a benefit over the $\mathcal{F}_r$ condition, which requires looking back and inspecting the set of $\mathbf{send}$ and $\mathbf{recv}$ actions for all message contents that appear in the trace (see example 5.4). In contrast to $\mathcal{F}_r$, no memory is needed for implementing $\mathcal{F}^\dagger$, and instead, the buffer $\mathcal{B}$ in the specification of $I^\dagger$ effectively encodes the history of traces as far as it is relevant to $\mathcal{F}^\dagger$.

Discussions

The concept of locally testable fairness constraints as is used here resembles *local fairness* of Alur and Henzinger [AH03] in the sense that these both strive to define fairness constraints which can be reduced to a local (per state) conflict resolution procedure. However, in [AH03],

a fairness constraint refers to two sets of transitions, such that if the first set occurs infinitely often in a fair trace, then the second set should also occur infinitely often. Roughly speaking, the notion of locality there requires that whenever a member of the first set is enabled at a state, then so is a member of the second set. In such cases, an unfair trace results only out of an unfair local conflict resolution algorithm, when both these sets are explicitly given. In contrast, our locality condition indicates that the members of the second set (the set whose members have to be treated fairly) can be determined only by local inspection.

Fairness constraints often concern infinite executions, that is, finite traces are by definition always fair (e.g. see [Fra86, CGP00]). Whereas, $\mathcal{F}_r$ and $\mathcal{F}^\dagger$ put conditions also on finite traces, hence being relevant even to finite acyclic LTSs. [11] The root of this difference lies in the $\mathcal{DY}$'s definition in § 5.2, which lacks the ability to remove messages from its knowledge. As traditionally the $\mathcal{DY}$ model has been used for verifying safety properties, the ability to remove messages has been considered irrelevant and, thus, ignored.

We give a faithful specification of $\mathcal{DY}$ in [CT06], in which the intruder can explicitly choose to terminate. With this capability, requiring maximum progress for finite traces does not force the intruder to do more than what is required by RCC (cf. example 5.1) and, therefore, maximal finite traces are also counted as RCC-fair. Here we choose not to follow the approach of [CT06], because the presentation and proofs would become so convoluted that the main ideas of the model get obscured. [12]

Prasad Sistla characterises *fairness properties* as those "insensitive to addition or deletion of prefixes" [Pra94]. According to this definition, $\mathcal{F}^\dagger$ qualifies as a fairness constraint, while $\mathcal{F}_r$ does not. We note that RCC is not a fairness property as well, as it is sensitive to prefixes. RCC is however an abstraction for the actual mechanisms that provide resilient channels. In fact, RCC would be unachievable in practice without imposing a fairness constraint on communication channels. See § 5.4 on realisation of resilient channels.

5.5.2 Equivalence of the two intruder models

For an arbitrary protocol $\mathcal{P}$, let $L = \partial_H(P \| \mathcal{DY}(\Gamma_0))$ and $L^\dagger = \partial_H(P \| I^\dagger(\Gamma_0, \emptyset)$. In this section, it is proved that for any liveness property ϕ we have

$$L \models_{\mathcal{F}_r} \phi \iff L^\dagger \models_{\mathcal{F}^\dagger} \phi.$$

This result hinges on certain assumptions: $\mathbf{recv}^\dagger$ is assumed not to appear as an action label in L and ϕ is assumed to *saturate* the equivalence relation $\sim_{\{\mathbf{recv}, \mathbf{recv}^\dagger\}}$ (defined below).

5.6. DEFINITION. *For $B \subseteq A$ and traces $\alpha = \alpha_1 \cdot \alpha_2 \cdots$ and $\alpha' = \alpha'_1 \cdot \alpha'_2 \cdots$ belonging to A^∞, we write $\alpha \sim_B \alpha'$ iff $\forall i.\ \alpha_i = \alpha'_i \vee (\alpha_i \in B \wedge \alpha'_i \in B)$.*

[11] Compare it to § 4.3 in [VV06], where a general definition of fairness constraints is proposed which indeed can put conditions on finite traces as well.

[12] In [CT06] we did not assume single-image LTSs. As a results we had to handle sets of sequences in mapping L and $L^\dagger$ and vice versa. The simple simulation relation used in § 5.5.2 would not then be sufficient and a variant of *ready trace equivalence* had to be used [Gla93].

Clearly, $\sim_B$ defines an equivalence relation.

5.7. DEFINITION. *A trace property $\phi \subseteq A^\infty$ saturates an equivalence relation iff for each equivalence class C of the relation, either $C \subseteq \phi$ or $\phi \cap C = \emptyset$.*

Intuitively, a property that saturates $\sim_B$ does not distinguish the elements of B. For example, in LTL terms (see definition 7.4), $\phi = \Diamond(\bar{1})$ and $\phi = \Diamond(\mathbf{recv} \vee \mathbf{recv}^\dagger)$ both saturate $\sim_{\{\mathbf{recv},\mathbf{recv}^\dagger\}}$, but $\Diamond(\mathbf{recv})$ does not. [13]

Observe that each state s in L is uniquely characterised by the pair (P, Γ), where P contains the states of honest processes of $\mathcal{P}$ along with their local variable bindings, and Γ is the knowledge set of $\mathcal{DY}$. To refer to the components of state s, we write $s.P$ and $s.\Gamma$. Each state $s^\dagger$ in $L^\dagger$ is uniquely characterised by the tuple $(P, \Gamma, \mathcal{B})$, where P contains the states of honest processes of $\mathcal{P}$ and Γ and $\mathcal{B}$ are the knowledge sets of $I^\dagger$. Similarly we write $s^\dagger.P$, $s^\dagger.\Gamma$ and $s^\dagger.\mathcal{B}$ to refer to the components of $s^\dagger$. Below, as a convention, the † superscript is used to refer to the elements of $L^\dagger$. Next, we define simulation for LTSs, modulo a relation on action labels. Below, $\lambda(B)$, with $B \subseteq Act$, denotes the set of action labels which appear in B (recall the notations of § 4.1).

5.8. DEFINITION. *For two LTSs $L_1 = (S_1, s_{01}, Act, Tr_1)$ and $L_2 = (S_2, s_{02}, Act, Tr_2)$, we say L_2* simulates *L_1 modulo a relation $\rho \subseteq \lambda(Act) \times \lambda(Act)$, denoted $L_1 \leq_\rho L_2$, iff there exists a relation $R \subseteq S_1 \times S_2$ such that:*

$$
\begin{array}{ll}
(i) & R(s_{01}, s_{02}). \\
(ii) & \forall s_1 \in S_1, s_2 \in S_2.\ R(s_1, s_2) \ \wedge\ s_1 \xrightarrow{a_1} s_1' \in Tr_1 \implies \\
 & \qquad \exists s_2' \in S_2, a_2 \in Act.\ s_2 \xrightarrow{a_2} s_2' \in Tr_2 \ \wedge\ R(s_1', s_2') \\
 & \qquad\qquad \wedge\ \rho(\lambda(a_1), \lambda(a_2)).
\end{array}
$$

To clarify the idea of simulation modulo a relation on action labels, we continue with a simple example. Let $p = send_p(\langle q, t\rangle) \cdot \delta$ and $q = recv_q(\langle q, t\rangle) \cdot q$ be two processes. Figure 5.3 shows LTSs corresponding to $L = \partial_H(p \| q \| \mathcal{DY}(\emptyset))$ and $L^\dagger = \partial_H(p \| q \| I^\dagger(\emptyset, \emptyset))$. The sets beside states show the content of $\Gamma, \mathcal{B}$ pair of $I^\dagger$ in those states, and similarly for $\mathcal{DY}$. We note that $L^\dagger \leq_\rho L$, with $\rho = \{(\mathbf{send}_p, \mathbf{send}_p), (\mathbf{recv}_p, \mathbf{recv}_p), (\mathbf{recv}_p^\dagger, \mathbf{recv}_p)\}$. The simulation relation R is depicted in figure 5.3 with dotted lines.

Below, via a few lemmas, we establish a close relation between L and $L^\dagger$.

5.2. LEMMA. *$L \leq_{\mathrm{f}} L^\dagger$, with* $\mathrm{f} = \{(\ell, \ell) \mid \ell \in \lambda(Act)\} \cup \{(\ell, \ell') \mid \exists q \in \mathcal{P}.\ \ell = \mathbf{recv}_q \wedge \ell' = \mathbf{recv}_q^\dagger\}$.

Proof: Construct R as $R(s, s^\dagger)$ iff $s.P = s^\dagger.P$ and $s.\Gamma = s^\dagger.\Gamma$. Obviously $R(s_0, s_0^\dagger)$. Now, assume $s \xrightarrow{a} s'$ and $R(s, s^\dagger)$. We distinguish three cases for a:

[13] We abuse the notation and define sets of actions based on their action labels. For instance, set $\{\mathbf{recv}\}$ stands for the set of all actions that have $\mathbf{recv}$ as action label, namely $\{a \in Act \mid \lambda(a) = \mathbf{recv}\}$.

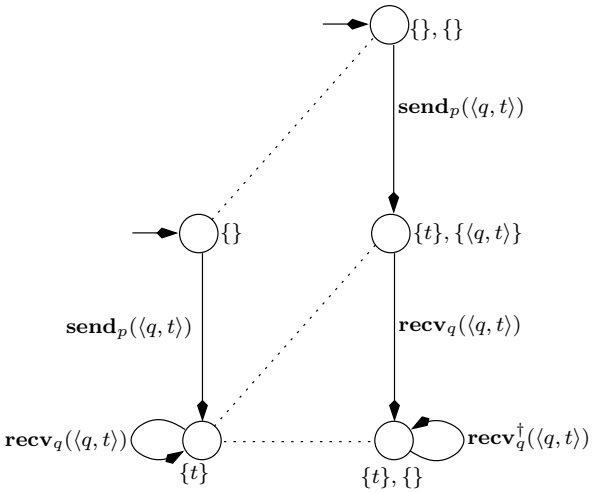


Figure 5.3: *Comparing LTSs resulting from $\mathcal{DY}$ (left) and $I^\dagger$ (right).*

- a is an internal action: It only depends on $s.P$ and only changes the state of the corresponding honest process(es). Therefore, a can faithfully be simulated in $L^\dagger$.
- $a = \mathbf{send}_p(\langle q, m\rangle)$: It only depends on $s.P$ and adds m to $s.\Gamma$. Clearly, $s^\dagger.P$ allows performing the $\mathbf{send}$ action and m is added to $s^\dagger.\Gamma$. Moreover, $\langle q, m\rangle$ is added to $s^\dagger.\mathcal{B}$, which does not affect the R relation.
- $a = \mathbf{recv}_q(\langle q, m\rangle)$: It depends on $s.P$ and $s.\Gamma$. We observe that $s.\Gamma = s^\dagger.\Gamma$ and, thus, $I^\dagger$ can also simulate $\mathcal{DY}$. The only difference is that if $\langle q, m\rangle \in s^\dagger.\mathcal{B}$, then the resulting action will be called $\mathbf{recv}^\dagger_q(\langle q, m\rangle)$. Note that since both $(\mathbf{recv}_p, \mathbf{recv}_p)$ and $(\mathbf{recv}_p, \mathbf{recv}^\dagger_p)$ belong to f, $I^\dagger$ can in any event simulate $\mathcal{DY}$.

This completes our proof. ■

5.3. LEMMA. *$L^\dagger \leq_{\mathrm{f}^{-1}} L$ with f^{-1} given as the inverse image of f, namely* $\mathrm{f}^{-1} = \{(\ell, \ell) \mid \ell \in \lambda(Act) \wedge \neg\exists q \in \mathcal{P}.\ \ell = \mathbf{recv}^\dagger_q\} \cup \{(\ell, \ell') \mid \exists q \in \mathcal{P}.\ \ell = \mathbf{recv}^\dagger_q \wedge \ell' = \mathbf{recv}_q\}$.

Proof: Construct R as $R(s^\dagger, s)$ iff $s^\dagger.P = s.P$ and $s^\dagger.\Gamma = s.\Gamma$. Obviously $R(s^\dagger_0, s_0)$. Now, assume $s^\dagger \xrightarrow{a^\dagger} s'^\dagger$ and $R(s^\dagger, s)$. Note that for any state $s^\dagger$, we have $\langle q, m\rangle \in s^\dagger.\mathcal{B} \implies \mathit{synth}(m, s^\dagger.\Gamma) = \mathsf{T}$. A case analysis of $a^\dagger$ (similar to the case analysis of lemma 5.2) completes the proof. ■

As we used the same simulation relation R in these lemmas, in the following we fix the notation and write $R(s, s^\dagger)$ iff $s.P = s^\dagger.P$ and $s.\Gamma = s^\dagger.\Gamma$.

5.4. LEMMA. *Any trace α in L can be uniquely translated to a trace $\alpha^\dagger$ in $L^\dagger$ such that $R(s_i^\alpha, s_i^{\alpha^\dagger})$, for all $i \geq 0$, and vice versa.*

Proof: Let $\alpha = \alpha_1 \cdots \alpha_i \cdots$ be a trace of L. As $L \leq_{\mathrm{f}} L^\dagger$, the sequence of states s^α corresponds to a sequence of states $s^\dagger$ in $L^\dagger$, such that $R(s_i^\alpha, s_i^\dagger)$ and $s_0^\dagger \xrightarrow{\alpha_1^\dagger} s_1^\dagger \xrightarrow{\alpha_2^\dagger} \cdots$ with $(\lambda(\alpha_i), \lambda(\alpha_i^\dagger)) \in \mathrm{f}$, for all $i \geq 0$. To show the uniqueness of the translation, we note that the set $F(\ell) = \{\ell' \mid (\ell, \ell') \in \mathrm{f}\}$ is a singleton, except when $\ell = \mathbf{recv}_q$, for some $q \in \mathcal{P}$. In this case, we have $F(\mathbf{recv}_q) = \{\mathbf{recv}_q, \mathbf{recv}_q^\dagger\}$. Let us assume that $\alpha_i = \mathbf{recv}_q(\langle q, m\rangle)$. Observe that only one of the elements of $\mathrm{f}(\alpha_i)$ would be enabled at $s_{i-1}^\dagger$. This is due to the definition of $I^\dagger$: When $synth(m, s_i^\dagger.\Gamma) = \mathsf{T}$, action $send_I^\dagger(\langle q, m\rangle)$ can be performed only if $\langle q, m\rangle \notin s^\dagger.\mathcal{B}$ (see § 5.5.1). This shows the uniqueness of the resulting $\alpha^\dagger$ in $L^\dagger$.

Translating traces from $L^\dagger$ to L works similarly (in fact, simpler, as f^{-1} is a function, in contrast to f which is a relation). ■

Let α and $\alpha^\dagger$ be traces respectively in L and $L^\dagger$. Lemma 5.4 states that α can be uniquely translated to $L^\dagger$. As a convention, we write $\mathrm{f}(\alpha)$ for the resulting trace. Trace $\mathrm{f}^{-1}(\alpha^\dagger)$ is defined similarly. Note that since ϕ saturates $\sim_{\{\mathbf{recv}, \mathbf{recv}^\dagger\}}$, we have $\alpha \in \phi \iff \mathrm{f}(\alpha) \in \phi$ and $\alpha^\dagger \in \phi \iff \mathrm{f}^{-1}(\alpha^\dagger) \in \phi$. The translation relations f and f^{-1} therefore preserve ϕ.

5.1. THEOREM. *For an arbitrary protocol $\mathcal{P}$ with a liveness property ϕ, the following holds*

$$\partial_H(P \| \mathcal{DY}(\Gamma_0) \models_{\mathcal{F}_r} \phi \iff \partial_H(P \| I^\dagger(\Gamma_0, \emptyset)) \models_{\mathcal{F}^\dagger} \phi$$

Proof: We show that if α is a trace in L such that $\neg\mathcal{F}_r(\alpha)$, then $\neg\mathcal{F}^\dagger(\mathrm{f}(\alpha))$, and vice versa. Since ϕ is preserved by f and f^{-1}, showing that fairness constraints agree, in the above sense, is enough to prove the theorem.

We start by assuming $\neg\mathcal{F}_r(\alpha)$, with α being a trace in L. For $s^{\mathrm{f}(\alpha)}$ (sequences of states corresponding to $\mathrm{f}(\alpha)$ in $L^\dagger$) we write $s^\dagger$. According to lemma 5.4, $\forall i \geq 0.\ R(s_i^\alpha, s_i^\dagger)$. Consider two cases for α: Finite and infinite. If α is finite, namely $\alpha = \alpha_1 \cdots \alpha_n$, then $\neg\mathcal{F}_r(\alpha)$ means that $en_T(s_n^\alpha)$ has at least one element (i.e. enabled transition) which does not belong to Θ_n^α. We claim that $en(s_n^\dagger)$ has an element (i.e. enabled action) which label is not $\mathbf{recv}^\dagger$. To see why, note that if the enabled transition of s_n^α has an action label different from $\mathbf{recv}$, then it is present at $s_n^\dagger$ as well (according to lemma 5.2), proving the claim. Now, assume that the enabled transition of s_n^α has $\mathbf{recv}_q(\langle q, m\rangle)$ as its action. Since it does not belong to Θ_n^α we conclude that the corresponding message belongs to $s_n^\dagger.\mathcal{B}$ (recall definition 5.3 introducing Θ). Therefore, $I^\dagger$ cannot do $send_I^\dagger(\langle q, m\rangle)$ and has to instead perform $send_I(\langle q, m\rangle)$, thus resulting in $\mathbf{recv}_q(\langle q, m\rangle)$ in $L^\dagger$. This proves the claim. Let us now assume that α is infinite. $\neg\mathcal{F}_r(\alpha)$ means that some transition not belonging to Θ^α is infinitely enabled, but not infinitely often performed. The same rationale as in the above argument shows that then an action not equal to $\mathbf{recv}^\dagger$ is infinitely often enabled in $L^\dagger$ on the run of $\mathrm{f}(\alpha)$, but not infinitely often performed by $\mathrm{f}(\alpha)$. Therefore, $\neg\mathcal{F}^\dagger(\mathrm{f}(\alpha))$.

Now assume $\neg\mathcal{F}^\dagger(\alpha^\dagger)$, with $\alpha^\dagger$ being a trace in $L^\dagger$. This means that $\alpha^\dagger$ is ignoring an action which label is not $\mathbf{recv}^\dagger$. A similar argument as above (in fact a simpler argument, since f^{-1} is a function) shows that this implies that $\mathrm{f}^{-1}(\alpha^\dagger)$ ignores a transition that does not belong to $\Theta^{\mathrm{f}^{-1}(\alpha^\dagger)}$, hence $\neg\mathcal{F}_r(\mathrm{f}^{-1}(\alpha^\dagger))$. ■

When checking safety properties, no fairness constraints are applied. The intuition behind this is that since attacks on safety properties are finite traces, the intruder does not gain anything by quitting the protocol. Whereas for subverting a liveness requirement the intruder may in principle by quitting the protocol prevent the honest parties from ever reaching a certain goal. When no fairness constraints are applied, the process $\mathcal{DY}$ is equivalent to process $I^\dagger$, after renaming $send_I^\dagger$ to $send_I$ actions. Hence, the LTSs resulting from the interactions between an arbitrary protocol with these two intruder models are isomorphic after renaming (see lemmas 5.2 and 5.3). The following theorem articulates this observation. Recall that ϕ, the property to be checked, saturates $\sim_{\{\mathbf{recv},\mathbf{recv}^\dagger\}}$.

5.2. THEOREM. *For an arbitrary protocol $\mathcal{P}$ with a safety property ϕ, the following holds*

$$\partial_H(P\|\mathcal{DY}(\Gamma_0)) \models \phi \iff \partial_H(P\|I^\dagger(\Gamma_0,\emptyset)) \models \phi$$

Since any trace property can be written as the intersection of a liveness property and a safety property [AS84], any security property which is a trace property can be checked using the $I^\dagger$ intruder model. We illustrate this in particular for FE properties in chapter 6.

5.5.3 A liveness property for FE

In this section we focus on a general liveness property which can in particular encode the requirements of FE protocols: When action $\mathtt{1}$ happens, reaching $\bar{\mathtt{1}}$ is inevitable. [14] For instance, the fairness requirement of FE protocols (see § 2.2) states that once A receives her desired item, denoted by action $\mathtt{1}$, B will inevitably receive his desired item, denoted by $\bar{\mathtt{1}}$. In short, $\mathtt{1}$ is inevitably followed by $\bar{\mathtt{1}}$.

FE protocols are concerned with protecting an honest party from possible malicious behaviour of the opponent. Therefore, when verifying FE, the intruder is a legitimate, though malicious, principal of the protocol. Note that the $\mathcal{F}_r$ and $\mathcal{F}^\dagger$ constraints do not rule out any (malicious) behaviour of the intruder as a principal. In particular, the intruder can prematurely abort the protocol as a principal, while continuing being the network. This is because sending any message by the intruder is deemed unnecessary in $\mathcal{F}_r$ and $\mathcal{F}^\dagger$ when the message is not buffered in resilient channels, even if the message is supposed to be sent by the legitimate role that the intruder plays in the protocol.

In § 5.5.2 we showed that verifying liveness properties with the RCC_1 assumption can be done using a modified intruder model and a simpler fairness constraint. Below, we use

[14] This property is sometimes called *responsiveness*, e.g. see [AS84].

the regular alternation-free fragment of μ-calculus (see § 4.2) to encode this liveness property under the $\mathcal{F}^\dagger$ fairness constraint:

$$\varphi = [\mathsf{T}^* \cdot \imath \cdot (\neg\bar{\imath})^*]\langle(\neg\mathbf{recv}^\dagger(\_))^* \cdot \bar{\imath}\rangle\mathsf{T}$$

The intuitive meaning of φ is, whenever $\imath$ has happened but $\bar{\imath}$ has not (yet) occurred, there is an execution path to $\bar{\imath}$ that does not contain any $\mathbf{recv}^\dagger$ actions. $\mathcal{DY}$ is thus not forced in such executions to collaborate more than delivering received messages. Formula φ is equivalent to ϕ under the fairness constraint $\mathcal{F}^\dagger$, where ϕ states the inevitable reachability of $\bar{\imath}$ after $\imath$, namely $\phi = [\mathsf{T}^* \cdot \imath]\ \mu X.\ (\langle\mathsf{T}\rangle\mathsf{T} \wedge [\neg\bar{\imath}]X)$, cf. [MS03]. The following example, which studies the protocols of examples 5.1 and 5.2 with the $I^\dagger$ intruder model, gives an intuitive explanation of φ.

5.5. EXAMPLE. *Recall example 5.1. Let $p = send_p(\langle q, t_1\rangle) \cdot \delta$ and $q = recv_q(\langle q, h(t_1)\rangle) \cdot \bar{\imath} \cdot \delta$. Here, we study $L^\dagger = \partial_H(p\|q\|I^\dagger(\emptyset,\emptyset))$. Let ϕ be the set of traces that contain $\bar{\imath}$. We implement ϕ under the $\mathcal{F}^\dagger$ constraint as $\varphi' = [(\neg\bar{\imath})^*]\langle(\neg\mathbf{recv}^\dagger(\_))^* \cdot \bar{\imath}\rangle\mathsf{T}$. Note the difference between φ and φ'. Figure 5.4 shows $L^\dagger$. As is evident in the LTS, the only trace sprouting out of s_0 which contains $\bar{\imath}$, also contains a $\mathbf{recv}^\dagger$ action in between. Consequently, $s_0 \not\models \varphi'$.*

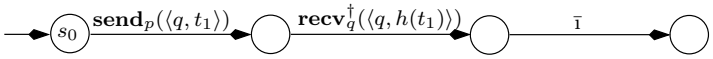


Figure 5.4: *LTS of the first protocol of example 5.5.*

Now we turn to example 5.2. Recall that $p = send_p(\langle q, t_1\rangle) \cdot p$ and $q = recv_q(\langle q, t_1\rangle) \cdot q + recv_q(\langle q,\ t_1, t_1\rangle) \cdot \bar{\imath} \cdot \delta$. Figure 5.5 depicts the LTS $L^\dagger = \partial_H(p\|q\|I^\dagger(\emptyset,\emptyset))$. Consider the property φ' as above. Observe that the only path to $\bar{\imath}$ from s_0 passes $\mathbf{recv}^\dagger_q(\langle q,\ t_1, t_1\rangle)$.

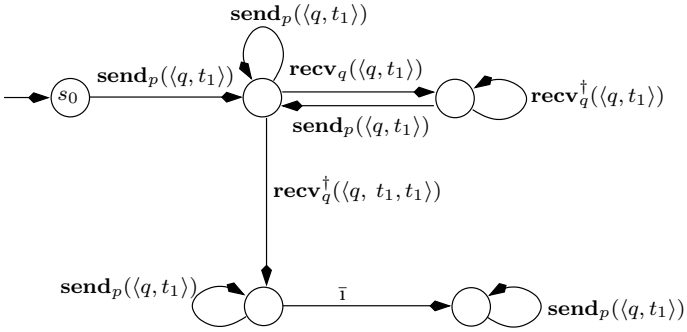


Figure 5.5: *LTS of the second protocol of example 5.5.*

Therefore $s_0 \not\models \varphi'$.

It is worth mentioning that in case sending messages which are not required by RCC would help the intruder to launch an attack on the protocol, $\mathcal{F}^\dagger$ does not prevent the intruder to do so. Conversely, if the messages buffered in resilient channels are all undeliverable, $\mathcal{F}_r$ allows the intruder to ignore them. The following two examples clarify these concepts.

5.6. EXAMPLE. *We consider processes* $p = send_p(\langle q, t_1\rangle) \cdot \delta$ *and* $q = recv_q(\langle q, t_1\rangle) \cdot \bar{1} \cdot \delta + recv_q(\langle q,\ t_1\rangle) \cdot recv_q(\langle q,\ t_1\rangle) \cdot \delta$. *Let* $L^\dagger = \partial_H(p \| q \| I^\dagger(\emptyset, \emptyset))$. *Figure 5.6 depicts* $L^\dagger$. *Consider the property* φ' *as in example 5.5. Observe that* $s_0 \not\models \varphi'$. *This is because* $[(\neg\bar{1})^*]$

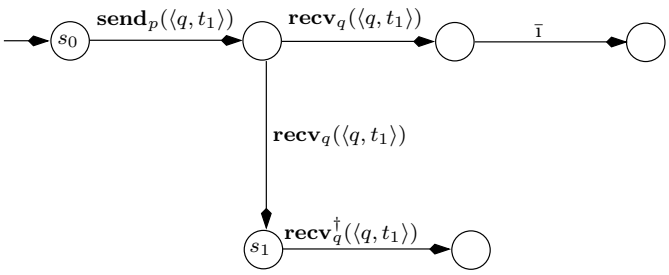


Figure 5.6: *LTS of the protocol of example 5.6.*

can take the system from s_0 *to* s_1, *from which no trace leads to* $\bar{1}$.

5.7. EXAMPLE. *Let two processes* p *and* q *be defined through the following equations:*

$$p = send_p(\langle q, h(t_1)\rangle) \cdot p_1$$
$$p_1 = send_p(\langle q, t_2\rangle) \cdot p_1$$
$$q = \Sigma_{v \in \{t_1, t_2\}} recv_q(\langle q, h(v)\rangle) \cdot recv_q(v) \cdot \bar{1} \cdot \delta,$$

with $t_1, t_2 \in \mathcal{T}$ *and* $t_1 \neq t_2$. *As in the previous examples, consider* $L^\dagger = \partial_H(p \| q \| I^\dagger(\emptyset, \emptyset))$ *and the liveness property* φ' *(defined in example 5.5). Figure 5.7 shows* $L^\dagger$. *Observe that* p *sends an infinite number of* t_2 *to* $I^\dagger$. *This is however simply ignored, since* t_2 *is undeliverable to* q. *Indeed,* $L \not\models \varphi'$. *Note that imposing a naïve fairness constraint such as "if a message is sent infinitely often to the resilient channel, it is infinitely often delivered", below referred to as* f_c, *would lead to absurd results. This is because for any maximal trace* $\alpha^\dagger$ *in* $L^\dagger$ *we have* $f_c(\alpha^\dagger) = \mathsf{F}$, *thus,* $f_c(\alpha^\dagger) \implies \phi(\alpha^\dagger)$ *evaluates to* T *for any* ϕ *(cf. example 5.2). Consequently,* $L^\dagger \models_{f_c} \phi$ *for any* ϕ. *Clearly,* f_c *is not suitable for formalising* RCC_1.

We observe that often only *some* of the channels in a communication network need to be resilient in order to achieve liveness (e.g. only channels to and from the TTP need to be resilient). [15] One way to implement this in $I^\dagger$ is to add a function ζ, characterising messages

[15] Similar protocol specific constraints can easily be expressed in our intruder model. For example, see the intruder specifications in the case studies of § 6.

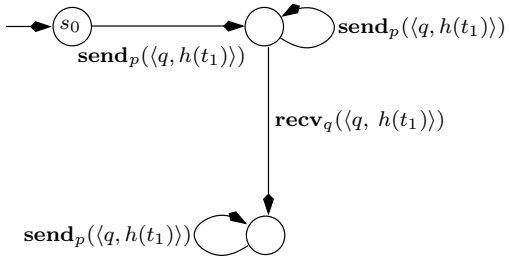


Figure 5.7: *LTS of the protocol of example 5.7.*

which should be added to the buffer (set $\mathcal{B}$) of resilient channels. The new specification of $I^\dagger$ is then (cf. § 5.5.1):

$$
\begin{aligned}
I^\dagger(\Gamma, \mathcal{B} : Set) = \quad & \Sigma_{q\in\mathcal{P}, m\in Msg} \\
& \qquad recv_I(\langle q, m\rangle) \cdot I^\dagger(\{m\} \bigcup \Gamma, \{\langle q, m\rangle\} \cup \mathcal{B}) \\
& \qquad\quad \lhd \zeta(\langle q, m\rangle) \rhd \\
& \qquad recv_I(\langle q, m\rangle) \cdot I^\dagger(\{m\} \bigcup \Gamma, \mathcal{B}) \\
& + \\
& \Sigma_{q\in\mathcal{P}, m\in Msg} \\
& \qquad send_I(\langle q, m\rangle) \cdot I^\dagger(\Gamma, \mathcal{B} \setminus \{\langle q, m\rangle\}) \lhd \langle q, m\rangle \in \mathcal{B} \rhd \delta \\
& + \\
& \Sigma_{q\in\mathcal{P}, m\in Msg} \\
& \qquad send^\dagger_I(\langle q, m\rangle) \cdot I^\dagger(\Gamma, \mathcal{B}) \lhd synth(m, \Gamma) \wedge \langle q, m\rangle \notin \mathcal{B} \rhd \delta
\end{aligned}
$$

The function ζ can for instance depend on contents of messages, their destination (e.g. messages destined to a trusted entity), etc. See [CT04] for an example of such a ζ function.

We conclude this section by briefly describing one of the most interesting attacks that has been reported on timeliness. The attack was discovered by Boyd and Kearney [BK00] on an optimistic fair certified email protocol, proposed by Zhou, Deng and Bao in [ZDB99] and [ZDB00]. In this protocol, Alice, the sender of the email, starts by sending a message to Bob, which contains her commitment not only to the email text, but also to a key which is used to encrypt the email (recall figure 2.1). At this early stage, Bob does not know the key used for encrypting the email, and thus cannot check whether Alice's commitment to the key is genuine or simply a random bit string. Alice can exploit this ignorance of Bob in the scenario described below.

Alice sends a random bit string to Bob instead of her commitment to the key. After receiving Bob's commitment (message 2 in figure 2.1), Alice stops the optimistic protocol. Bob cannot resolve to the TTP, simply because the TTP needs Alice's commitments; and to the TTP it is not clear whether Bob is sending a bogus message as Alice's commitment to the key, or Alice has sent a bogus message to Bob. On the other hand, Alice can simply resolve the

exchange whenever she wishes, because she can compose and send a genuine key commitment to the TTP, and resolve the exchange. Only after Alice has resolved the exchange, the TTP allows Bob to resolve (in the protocol of Zhou et al., the TTP treats previously resolved exchanges differently from the exchanges which has never been disputed).

In this scenario, Bob can terminate the exchange only with Alice's help. Therefore the property φ is violated in this protocol, not because $\bar{\imath}$ is unreachable, but because it is reachable only via the actions of the intruder (i.e. Alice). Note that when Alice terminates the exchange, Bob can terminate it as well. Therefore, fairness is not violated. The flaw described above reflects the fact that Bob cannot unilaterally terminate the exchange.

5.6 Related work

Below, we review some notable work in formal analysis of (liveness aspects of) FE protocols and compare them to our method.

Based on a decision procedure presented in [KKW05], Kähler and Küsters propose a constraint solving approach to analyse finite rounds of contract signing protocols, while the intruder is provided with unbounded fresh data [KK05]. Instead of resilient channels, they use synchronous authenticated confidential channels, thus restricting the intruder's abilities. Protocol participants are modelled as acyclic non-replicating processes. The liveness aspects of FE requirements then naturally disappear and, under certain conditions, the resulting safety properties are shown to be decidable. Had the intruder been given the power to delay messages over resilient channels, a fairness constraint similar to $\mathcal{F}_r$'s condition on finite traces would have been necessary for the model of [KK05]. In comparison, our results are stated for an arbitrary protocol that can possibly contain cyclic participants, provided that the interaction between the protocol and $\mathcal{DY}$ results in a finite state space (see below). Our analysis method, as well as the finite state analysis techniques discussed below, do not detect type-flaw attacks (see § 4.3). [16] Remarkably, it is possible to detect these attacks in [KK05].

The results of [KK05] are extended by Kähler, Küsters and Truderung, with proving the decidability of a large fragment of alternating-time μ-calculus on the infinite models induced by acyclic non-replicating security processes [KKT07]. [17] This fragment of alternating time μ-calculus notably comprises the game-based security properties of FE as formalised by Kremer and Raskin (see below). In the formalisation of [KKT07], resilient channels are modelled as a fairness constraint, used as a precondition for protocol requirements: They consider formulae of the form $f_c \implies \phi$, with f_c being the fairness constraint that guarantees eventual

[16] We note that in our approach it is possible to detect *simple* type-flaw attacks, where an atomic term of a certain type is confused with an atomic term of another type. However, *compound* type-flaws, where a composition of several terms is accepted as a term of another type, are not detectable in our method.

[17] In the formalisation of [KKT07], the participants are allowed though to loop only as a means to mimic the situation where they do not have any other action to perform. Such loops thus merely let a deadlock state to stutter, without consuming or generating messages.

message delivery over resilient channels and, ϕ the goal of the protocol. The constraint f_c requires that any resilient channel which buffers messages is eventually emptied (see § 5 in [KKT07]). Notice that if a message which is undeliverable (e.g. it has no recipient) is sent over resilient channels, then in any maximal trace, f_c is violated, cf. example 5.7. In such situations, $f_c \implies \phi$ would evaluate to T for any ϕ. To avoid such absurd results, an exact formalisation of the assumptions based on which protocols are modelled or using a fine tuned fairness constraint (such as $\mathcal{F}_r$) in [KKT07] seem to be necessary.

Moreover, [KKT07] shows that when the participant processes are allowed to have idle cycles (i.e. self-loops in which they do not consume or generate messages), then the game-based security properties are not decidable, cf. [HS04]. See also § 4.3.

We aim at automatic verification of FE protocols with cyclic specifications. Finiteness of the LTSs resulting from the interaction between a protocol and the intruder models is thus required in our analysis (see § 4.3). This poses some limitations: First, only a finite number of *actual* protocol sessions can be considered. To be precise, protocol participants cannot be provided with unbounded sources of fresh data and cannot replicate. They can only run a finite number of protocol sessions and then either terminate or turn into puppet participants, which loop without generating new data. This restriction cannot be easily lifted in automatic verification techniques, since security of protocols in general is undecidable (see § 2.1.1). Second, if there are processes in the protocol with loops in their specifications, then the intruder cannot in general be provided with unbounded sources of new data (such as fresh nonces). In fact, trusted parties are often looping participants that perpetually resolve incoming requests in optimistic FE protocols [Aso98]. As another example, a vendor selling a finite number of items, that waits for a purchase request, handles it and recurs, can also be modelled as a looping participant. The type-correct assumption is also generally required in our analyses to keep the state space finite. See also § 4.3.

In finite state analysis, the liveness aspects of FE have sometimes been left out from formal verification, e.g. see [SM02, GR03, DM05]. There are however notable exceptions [KR01, CCT05, ACC07]. Kremer and Raskin present a game-based semantics for FE properties, which can neatly expresses the desired behaviours of FE protocols [KR01]. The verification of FE requirements is then reduced to certain strategy problems. For instance, fairness for Alice can be expressed as "a coalition of Bob and [non-resilient] channels does not have a strategy to obtain [email content $\mathcal{M}$] without Alice having a strategy obtain a non-repudiation of receipt evidence" [KR01]. To exclude unrealistic behaviours of the model, a fairness constraint similar to $\mathcal{F}_0$ is put on the transition system in [KR01]. Their formalisation thus in principle suffers from the problems demonstrated in examples 5.1 and 5.2. Moreover, the $\mathcal{DY}$ intruder in [KR01] (which is combined with all corrupted insider parties) is specified *constructively*, i.e. a finite number of messages are presumed (or manually anticipated) as the set of messages that the intruder can receive, compose, etc. This formalisation clearly does not capture the infinite behaviour of the intruder. As their intruder's specification can only keep track of a finite pre-determined set of messages, it is dependent on the particular

protocol being analysed. This is in contrast to our generic recursive definition of $\mathcal{DY}$.

In [CCT05] we report an implementation of the intruder model $I^\dagger$ for analysing a fair non-repudiation protocol.

Armando, Carbone and Compagna use LTL for specifying protocol requirements and fairness constraints [ACC07]. Given a security protocol model L, requirement ϕ and fairness constraint f_c, they check $L \models (f_c \implies \phi)$. Their fairness constraints capture both the RCC-conforming behaviour of $\mathcal{DY}$ and the maximal progress of the honest processes. The authors also give an interesting overview on why some of the assumptions commonly made when analysing security protocols may fail when checking liveness properties. The fairness constraints used in [ACC07], however, strongly depend on the protocol being analysed and, thus, need to be devised for each protocol separately. Besides, [ACC07] provides no formal justification of these constraints. For instance, to model resilient channels as a fairness constraint, they require that each submitted message, should be delivered. This condition serves as the antecedent of the goals the protocol needs to achieve, i.e. as f_c in the aforementioned formula. It is easy to see that if a (malicious) process submits a message which is not deliverable (e.g. no process would receive it), then $f_c \implies \phi$ evaluates to T for any ϕ (see also example 5.7). To avoid such absurd results, a formal exposition of the assumptions based on which the approach is devised seems to be necessary.

In the context of machine-checked proofs, Wei and Heather have mechanised Schneider's verification [Sch98] of Zhou-Gollmann non-repudiation protocol [ZG96b] in PVS [WH07]. They use the *stable failures* semantics of CSP to reason about liveness. Intuitively, the stable failure semantics of process p consists of pairs $(\alpha, failure_\alpha)$, where α is a trace of p which cannot be extended by any internal actions, and the set $failure_\alpha$ contains the actions that p can refuse after executing α. To model the liveness property that action $\bar{\imath}$ inevitably occurs, using this semantics, [WH07] requires that any trace α either contains $\bar{\imath}$, or $\bar{\imath}$ does not belong to $failure_\alpha$ for all $(\alpha, failure_\alpha)$. We note that this approach suffers from the problem demonstrated in example 5.1, namely $\bar{\imath}$ in this example does not belong to any $failure_\alpha$, and yet the protocol does not guarantee the liveness property in the presence of an RCC-complying $\mathcal{DY}$.

In formalising RCC, we have essentially determined scheduling abilities of the intruder. Cortier, Küsters and Warinschi have a similar concern in [CKW07], chiefly in the computational model of cryptography. They argue why only fair scheduling of messages, à la [BPSW02], is not enough, and maximal progress of the processes involved in FE should also be guaranteed by fairness constraints. The model of [CKW07] contains a scheduler, apart from the intruder, while, to achieve finiteness without sacrificing inductive definitions, in our model the scheduler is the intruder. These have however the same goal, namely limiting the intruder's abilities just enough to impose RCC.

5.7 Conclusions

In this chapter we studied some concurrency issues regarding the Dolev-Yao intruder model which arise when verifying liveness properties. A liveness property does not hold in general in this model without the resilient communication channels assumption. We have given a formalisation of RCC, whose complexity indicates that the standard $\mathcal{DY}$ intruder, obliged to respect RCC, is not suitable for automatic verification. We have thus proposed a modified $\mathcal{DY}$ intruder model that respects RCC and fits in the existing verification frameworks. The fact that the proposed intruder model can be implemented in a general purpose process algebra equips us with already well developed tools and techniques for modelling and verification. In § 6, this intruder model is used to verify two FE protocols.

Chapter 6

Verifying liveness: Case studies

6.1 Introduction

This chapter reports on two verification case studies using the intruder model of § 5.

In § 6.2, we verify a fair payment protocol proposed in [VPG01]. We observe that the protocol does not achieve timeliness. As our intruder model is geared towards verifying liveness, detecting this timeliness flaw serves as an empirical basis for the effectiveness of the model. This case study appears first in the chapter and, thus, is described in great detail.

A fair digital rights management scheme, originally described in [NPG+05] and further extended in [TKJ07], is studied in § 6.3. This scheme heavily relies on trusted computing devices. We describe how our formal specification captures the characteristics of trusted devices. We also discuss how the verification phase has to accordingly be adapted to these changes in the specification. The core of the analysis technique nevertheless remains the same for the cases studies of § 6.2 and § 6.3.

Related work

Formal verification of security protocols in general is discussed in § 4.3. Formal verification of FE protocols, and in particular their liveness aspects, is reviewed in § 5.6. Formal verification of *functionality* of electronic commerce protocols, i.e. with no malicious party involved, can perhaps be traced back to [And92]. See also [HTWC96, KS03]. Further related work specific to each case study appears in its corresponding section.

6.2 Analysing a fair payment protocol

Vogt, Pagnia and Gärtner describe a protocol for fair exchange of *time sensitive* data [VPG01]. Time sensitive, in this context, refers to the information that loses its value over time, such as stock exchange quotes and location dependent information in mobile services. We note that when exchanging time sensitive data, a mere *eventual* delivery is not satisfactory, cf. § 2.2.

To enable the customers to roll back the exchanges which are excessively delayed, the protocol of [VPG01] assumes that each customer is equipped with a trusted computing device, e.g. a smart card. The protocol is intended for mobile applications, where "a vendor sells

a digital [time sensitive item] to a mobile customer who pays for it electronically" [VPG01]. Below, we describe this protocol. [1]

6.2.1 Protocol description

The protocol aims at fair exchange of time sensitive data for some amount of money, between a customer (C) and a vendor (V). The exchange uses a bank (B) as an on-line trusted entity [2] who manages the payment system. The payment system used in the protocol is assumed to support *revocable payments*, meaning that B has the power to roll back any payment transaction if needed, cf. § 2.2.2. Besides, a tamper-proof smart card (S) is attached to C. It is only through C that S can communicate with other parties.

It is assumed that cryptographic mechanisms are used to protect the integrity, authenticity and confidentiality of messages. Each party has therefore a resilient secure channel to communicate with S and B. Consequently, "eavesdropping, replay, and forging of messages are assumed to be impossible" [VPG01]. In case the attacker is the owner of S, namely C, it can permanently or temporarily disconnect S. Otherwise, S is tamper-proof.

The protocol starts when C wishes to purchase item d, offered by V, for the amount a. [3] A description $h(d)$ of d is assumed to be publicly known. A description function can for instance be a certified hash value of the content. We refer to [PVG03] for various solutions to secure item description and validation in FE. Note that the item d is confidential and, thus, should not be revealed to C unless it pays for it.

Below we describe the intended scenario of the protocol when all the participants are honest. This scenario is shown in protocol 6.1. At step 1, C sends a query to S for buying the item, which is described by $h(d)$ (namely d), for amount a. At step 2, S sends this request to V, possibly putting its signature on the payment order, the issuer's identity (C) and the identity of the beneficiary (V). At step 3, if a is the amount for which V wants to sell d, it sends a to B. If a contains a genuine payment order from C (which B can establish via checking, e.g., C and S's digital signatures in a) and C has the amount a in its account, then B transfers the amount a from C's account to V's. Note that because of the secure channels assumption made above, a cannot be replayed to maliciously charge C with this amount again. In practice, a should carry a fresh serial number and so forth. Afterwards, at step 4, B informs V that the money has successfully been transferred. At step 5, V acknowledges the payment by sending d to S. Note that the confidentiality assumption, mentioned above, implies that C cannot extract d from message 5, e.g. because d is sent encrypted with the public key of S. After receiving this message, S makes sure that d, received from V, matches $h(d)$, received from C.

[1] A minimal-delay variant of the protocol is also presented in [VPG01], which is not covered in our study. This variant is nonetheless susceptible to the timeliness flaw that we discuss here.

[2] If the payment mechanism allows off-line validation of payments, then B may be off-line.

[3] Our exposition of the protocol is slightly different from [VPG01]. There, an agreement between C and V on the item and its price happens outside the protocol. Whereas, here, we make this step explicit.

$$
\begin{array}{ll}
1. & C \rightarrow S:\ a, h(d), V \\
2. & S \rightarrow V:\ a, h(d), V \\
3. & V \rightarrow B:\ a, V \\
4. & B \rightarrow V:\ \mathit{transfer}(a) \\
5. & V \rightarrow S:\ d \\
 & S \text{ matches } d \text{ against } h(d) \\
6. & S \rightarrow C:\ h(d)? \\
7. & C \rightarrow S:\ \text{yes} \\
8. & S \rightarrow C:\ d
\end{array}
\tag{6.1}
$$

At step 6, S asks C whether the item described by $h(d)$, namely d, is still desirable for C or not. Since the protocol addresses exchanging time sensitive items, this step is of crucial importance, as it allows C to reject the exchange, in case the exchange has been excessively delayed. Note that if C agrees with receiving the item at step 7, by answering *yes* to S, the transfer of d from S to C happens locally, i.e. no further communications over open networks are needed. Therefore, if at step 7, d is still of value to C, so will it be at step 8. [4]

Some alternative scenarios are possible in the protocol. If C does not have the required amount of money a in its account, or if V does not want to sell d for the amount a, then S receives $\neg ack(a)$, and will subsequently inform C of the result:

$$
\begin{array}{ll}
4'. & B \rightarrow V:\ \neg \mathit{transfer}(a) \\
5'. & V \rightarrow S:\ \neg ack(a) \\
6'. & S \rightarrow C:\ \neg h(d)
\end{array}
\tag{6.2}
$$

In this case, S contacts B to *abort* the exchange (protocol 6.3, below). This is necessary, because otherwise a malicious V could simply send $\neg ack(a)$ to S, even after B transfers the amount a to V's account.

Similarly, C may answer *no* to the question $h(d)?$ at step 7. Before step 7, C has also the choice to send the message *no* to S to cancel the exchange (e.g. if C does not want to wait till S asks $h(d)?$). In all these cases, S will contact B to revoke the payment (if it has been transferred to V's account). This is called *aborting* the exchange:

$$
1^a. \quad S \rightarrow B:\ \neg a, V \tag{6.3}
$$

After receiving this message, B will never transfer a to V's account. In case a has already been transferred to V's account, B revokes the payment. According to the resilient channels assumption, this message will eventually reach B. Therefore, if a time sensitive item arrives late, C can drop the item and eventually get its money back (via S and B). We also remark that, although from V's point of view this protocol is not optimistic (as V has to contact B

[4] We remark that we have not left out any details of the protocol as it appears in [VPG01]. Unfortunately, the protocol even in its original form is underspecified and only verbally described.

in each exchange), the customer C needs to contact B only when a purchased item does not arrive in time.

Observe that V has no guarantees about the termination of the protocol, as at any moment a can be revoked. This is because, at step 7, C may arbitrarily delay answering S. Therefore, unless a payment is revoked, V cannot be sure about C's answer (*yes* or *no* to S) in the exchange. In other words, the protocol does not provide timeliness for V (cf. § 2.2), although this is one of the design goals according to [VPG01]. In our formal analysis, described below, this flaw is indeed detected.

6.2.2 Formal analysis

In this section we describe the steps followed to formally verify the protocol presented in § 6.2.1. Our approach is based on finite-state model checking, which (usually) requires negligible human intervention and, moreover, produces concrete counterexamples, i.e. attack traces, if the protocol fails to satisfy a desired property. However, a complete security proof of the protocol cannot, in general, be established by model checking. For an overview on formal methods for verifying FE protocols, see § 4.3. We first specify the protocol and the intruder model in the μCRL process algebraic language, see § 4.1. The μCRL tool-set can symbolically reduce and automatically generate LTSs associated to specifications, cf. § 4.1.2.

Second, we state the properties effectiveness, timeliness and fairness (see § 2.2), which are mentioned as the design goals of the protocol in [VPG01], in the regular alternation-free μ-calculus, see § 4.2. Finally, we check the protocol model with regard to the properties using the model checker EVALUATOR 3.0 from the CADP tool-set [FGK+96]. Below, these steps are described in detail.

Formal specification of the protocol

In FE protocols, besides protection from external intruders, each honest participant needs to be protected from his or her interlocutor as well. In our formal models, we consider two versions of each participant: Honest and malicious. An honest participant faithfully follows the protocol. To model a malicious participant we let the $\mathcal{DY}$ intruder process entirely control the participant. Our analyses thus have three cases: (i) both C and V behave according to the protocol, (ii) C is malicious, while V is honest and (iii) V is malicious, while C is honest. The case where both C and V are malicious players is not of our interest, as the protocol has to protect only honest players: A player who deliberately reveals its secret keys can hardly be protected. Note that S and B are assumed trusted and, thus, always follow the protocol.

We start with specifying the honest versions of C and V. [5] The specifications of B and S follow these. Since channels are authenticated, participants can always check the

[5] Here, we give the specifications of honest processes for one protocol round. This is enough to reveal the timeliness flaw of the protocol and, moreover, gives a more clear view of the idea behind the protocol and also the attack. In [CT04] we associate nonces to each exchange and, thus, scale up these specifications to multiple rounds.

origin of each message. Therefore, we extend the notation of § 5 so that each message is tagged with both the identity of its intended recipient and the identity of its genuine sender. We write $send_{A\to B}(m)$ and $recv_{A\leftarrow B}(m)$ as shorthands for $send_A(\langle B, A,\ m\rangle)$ and $recv_A(\langle A, B, m\rangle)$, respectively. In the specifications, instead of parentheses we use indentation to clarify bindings.

$$
\begin{array}{l}
C = \Sigma_{d\in Item, a\in Price, v\in \mathcal{P}_v} \\
\quad send_{C\to S}(a, h(d), p)\cdot \\
\quad\quad recv_{C\leftarrow S}(h(d)?)\cdot \\
\quad\quad \Sigma_{t\in\{\text{yes,no}\}} \\
\quad\quad\quad send_{C\to S}(t)\cdot\delta \\
\quad\quad\quad \lhd t = \text{no} \rhd \\
\quad\quad\quad send_{C\to S}(t)\cdot \\
\quad\quad\quad recv_{C\leftarrow S}(d)\cdot\delta \\
\quad + \\
\quad\quad recv_{C\leftarrow S}(\neg h(d))\cdot\delta \\
\quad + \\
\quad\quad send_{C\to S}(\text{no})\cdot\delta
\end{array}
$$

In C's specification, $Item$ is the set of items available in our model and $Price$ is the set of money orders that C can issue. Here, we assume $Item = \{d_1, d_2\}$ and $Price = \{a_1, a_2\}$. Since the criteria based on which C decides whether an item has arrived timely or not fall out of the scope of the protocol, in our specification these possibilities appear as a non-deterministic choice between *yes* and *no*. The set $\mathcal{P}_v$ consists of vendor identities, which we instantiate with $\mathcal{P}_v = \{V\}$. As a side remark, observe that $recv_C(\langle C,\ d\rangle)$ only accepts the item to which C has referred in message 1 of protocol 6.1.

$$
\begin{array}{l}
V = \Sigma_{d\in Item, a\in Price} \\
\quad recv_{V\leftarrow S}(a, h(d), V)\cdot \\
\quad\quad send_{V\to B}(a, V)\cdot \\
\quad\quad\quad recv_{V\leftarrow B}(transfer(a))\cdot send_{V\to S}(d)\cdot\delta \\
\quad\quad + \\
\quad\quad\quad recv_{V\leftarrow B}(\neg transfer(a))\cdot send_{V\to S}(\neg ack(a))\cdot\delta \\
\quad + \\
\quad\quad send_{V\to S}(\neg ack(a))\cdot\delta
\end{array}
$$

The bank B is specified as a cyclic process which perpetually resolves the disputes brought up by S and answers V's money transfer requests. The action $revoke_B(a, p)$ is performed to model revoking the transaction which transferred a to p's account (if it ever

happened in the past). The details of this procedure depend on the payment system used (e.g. see [JY96, Vog03]) and are thus abstracted away in our model. Abstracting away the payment system implies that we cannot model the situation where C does not have enough credits at B. This can however be simulated by assuming that V rejects such requests after privately consulting B (note that V has the choice to simply reject a purchase request).

In the specification of S, for clarity, we model the abort protocol as a separate process.

$$
\begin{aligned}
B(U : Set) = \Sigma_{a \in Price, p \in \mathcal{P}_v} \\
& recv_{B \leftarrow p}(a, p) \cdot \\
& \quad send_{B \rightarrow p}(\neg transfer(a)) \cdot B(U) \\
& \quad \lhd (a, p) \in U \rhd \\
& \quad send_{B \rightarrow p}(transfer(a)) \cdot B(U) \\
& + \\
& recv_{B \leftarrow S}(\neg a, p) \cdot revoke_B(a, p) \cdot B((a, p) \cup U)
\end{aligned}
$$

$$
\begin{aligned}
S = \Sigma_{a \in Price, d \in Item, p \in \mathcal{P}_v} \\
& recv_{S \leftarrow C}(a, h(d), p) \cdot \\
& send_{S \rightarrow p}(a, h(d), p) \cdot \\
& \quad recv_{S \leftarrow p}(ack(a)) \cdot \\
& \quad send_{S \rightarrow C}(h(d)?) \cdot \\
& \quad\quad recv_{S \leftarrow C}(\text{yes}) \cdot send_{S \rightarrow C}(d) \cdot \delta \\
& \quad + \\
& \quad\quad recv_{S \leftarrow C}(\text{no}) \cdot abort(a, p) \\
& + \\
& \quad recv_{S \leftarrow p}(\neg ack(a)) \cdot send_{S \rightarrow C}(\neg h(d)) \cdot abort(a, p) \\
& + \\
& \quad recv_{S \leftarrow C}(\text{no}) \cdot abort(a, p) \\
\\
abort(a, p : Msg) = send_{S \rightarrow B}(\neg a, p) \cdot \delta
\end{aligned}
$$

Our formalisation contains a $\mathcal{DY}$ intruder which does not disrupt resilient channels. Therefore, all messages exchanged in the protocol are guaranteed to be eventually delivered to their destination. Since liveness properties are part of FE requirements, we use the intruder model $I^\dagger$, defined in § 5.5.1, along with $\mathcal{F}^\dagger$ as the fairness constraint. See § 5.

The intruder, as discussed above, is not separated from malicious participants. To reflect the authenticated and secure channels assumption, the intruder is not able to comprehend messages not destined to it and, moreover, it cannot send a message with a sender tag different

from its own identity (and the identities of the processes that it has corrupted). It can however schedule all the messages exchanged between the participants of the protocol.

In scenario (ii), i.e. malicious C and honest V, the intruder is specified by instantiating $\mathsf{id} = C$ in algorithm 6.1 (cf. § 5.5.1). The intruder model for scenario (iii) can likewise be derived by letting $\mathsf{id} = V$. In scenario (i), the intruder is assumed to be an outsider, i.e. none of the players of the protocol are corrupted. This corresponds to $\mathsf{id} = \mathcal{DY}$, where $\mathcal{DY} \notin \{C, V, S, B\}$, in algorithm 6.1. Below, we give the formal specification of these scenarios (see the notations of § 4.1.2 and § 5).

- Scenario (i): $L_{(i)} = \partial_H(C \| V \| B \| S \| I^\dagger_{\mathcal{DY}}(\Gamma_{(i)}, \emptyset))$, where $\Gamma_{(i)} = \{C, V, S, B, \mathcal{DY}\} \cup \{h(d) \mid d \in \mathit{Item}\} \cup \{\text{yes}, \text{no}\}$.
- Scenario (ii): $L_{(ii)} = \partial_H(V \| B \| S \| I^\dagger_C(\Gamma_{(ii)}, \emptyset))$, where $\Gamma_{(ii)} = \Gamma_{(i)} \cup \mathit{Price}$.
- Scenario (iii): $L_{(iii)} = \partial_H(C \| B \| S \| I^\dagger_V(\Gamma_{(iii)}, \emptyset))$, where $\Gamma_{(iii)} = \Gamma_{(i)} \cup \mathit{Item}$.

$$
\begin{aligned}
I^\dagger_{\mathsf{id}}(\Gamma, \mathcal{B} : \mathit{Set}) = \quad & \Sigma_{p,q \in \mathcal{P}, m \in \mathit{Msg}} \\
& \quad \mathit{recv}_I(p,q,m) \cdot I^\dagger(\{m\} \bigcup \Gamma, \{(p,q,m)\} \cup \mathcal{B}) \\
& \quad \lhd p = \mathsf{id} \rhd \\
& \quad \mathit{recv}_I(p,q,m) \cdot I^\dagger(\Gamma, \{(p,q,m)\} \cup \mathcal{B}) \\
& + \\
& \Sigma_{p,q \in \mathcal{P}, m \in \mathit{Msg}} \\
& \quad \mathit{send}_I(p,q,m) \cdot I^\dagger(\Gamma, \mathcal{B} \setminus \{(p,q,m)\}) \lhd (p,q,m) \in \mathcal{B} \rhd \delta \\
& + \\
& \Sigma_{p,q \in \mathcal{P}, m \in \mathit{Msg}} \\
& \quad \mathit{send}^\dagger_I(p,q,m) \cdot I^\dagger(\Gamma, \mathcal{B}) \\
& \quad \lhd q = \mathsf{id} \wedge \mathit{synth}(m, \Gamma) \wedge (p,q,m) \notin \mathcal{B} \rhd \delta
\end{aligned}
$$

Algorithm 6.1: *The intruder process, subsuming corrupted parties*

Properties

In the following, we discuss and formalise the design goals of the protocol via a few lemmas and theorems. To prove these in our models, we use finite state model checking of alternation-free regular μ-calculus formulae (see § 4.2). In verifying effectiveness, scenario (i) is considered, while timeliness and fairness are verified for the scenarios (ii) and (iii). First come a few auxiliary lemmas.

A (malicious) customer C could possibly get hold of an item d by other means than from S in action $\mathbf{send}_{S \to C}(d)$. To show that this is not the case, we verify the following lemma.

6.1. LEMMA. *$L_{(i)}$ and $L_{(ii)}$ satisfy the following property for any $d \in Item$:*

$$[(\neg \mathbf{send}_{S\to C}(d))^*.\mathrm{reveal}(d)]\mathsf{F},$$

where $\mathrm{reveal}(d)$ *is an internal action that the intruder performs when it learns d. We can model this by adding the following alternative choice to the intruder process in algorithm 6.1:* $\Sigma_{d\in Item}\mathrm{reveal}(d).I^{\dagger}_{\mathsf{id}}(\Gamma,\mathcal{B}) \lhd \mathit{synth}(d,\Gamma) \rhd \delta$.

6.2. LEMMA. *$L_{(i)}$ and $L_{(iii)}$ satisfy the following properties for any $d \in Item$:*

$$[\mathsf{T}^* \cdot \mathbf{send}_{C\to S}(a, h(d), V) \cdot \mathsf{T}^* \cdot \mathbf{send}_{S\to C}(d) \cdot \mathsf{T}^* \cdot \mathbf{send}_{S\to B}(\neg a, V)]\mathsf{F}$$
$$[\mathsf{T}^* \cdot \mathbf{send}_{C\to S}(a, h(d), V) \cdot \mathsf{T}^* \cdot \mathbf{send}_{S\to B}(\neg a, V) \cdot \mathsf{T}^* \cdot \mathbf{send}_{S\to C}(d)]\mathsf{F}$$

And $L_{(ii)}$ satisfies the following properties:

$$[\mathsf{T}^* \cdot \mathbf{recv}^{\dagger}_{S\leftarrow C}(a, h(d), V) \cdot \mathsf{T}^* \cdot \mathbf{send}_{S\to C}(d) \cdot \mathsf{T}^* \cdot \mathbf{send}_{S\to B}(\neg a, V)]\mathsf{F}$$
$$[\mathsf{T}^* \cdot \mathbf{recv}^{\dagger}_{S\leftarrow C}(a, h(d), V) \cdot \mathsf{T}^* \cdot \mathbf{send}_{S\to B}(\neg a, V) \cdot \mathsf{T}^* \cdot \mathbf{send}_{S\to C}(d)]\mathsf{F}$$

This lemma states that S, the smart card, never revokes the payment for a content which has been revealed to C. Lemmas 6.1 and 6.2 imply that if C learns an item, the associated payment is never revoked by S. The fact that B does not revoke any payment unless S requests it, is established with the following lemma.

6.3. LEMMA. *$L_{(i)}$, $L_{(ii)}$ and $L_{(iii)}$ satisfy the following property for any $a \in Price$ and $p \in \mathcal{P}_v$:*

$$[(\neg \mathbf{send}_{S\to B}(\neg a, p))^* \cdot \mathit{revoke}_B(a, p)]\mathsf{F}$$

Effectiveness (cf. § 2.2): According to Asokan et al. [ASW98b], when checking effectiveness, we assume that the "two honest players [C and V] interact directly with each other. The adversary in this case can interact with [trusted entities], but cannot interfere with the interaction of [C and V], except insofar as the adversary still has the power to schedule both [C's and V's] interaction with [trusted entities]". Effectiveness then means that "it is infeasible for the adversary ... to prevent [C and V] from successfully exchanging [their items]", when "neither players 'times out'". To model this scenario we use $L_{(i)}$, except that in $L_{(i)}$ C may send *no* to S, and V may not want to sell d for a, while C and V are assumed not to "time out". Therefore, below, we explicitly exclude these time out cases when checking effectiveness. [6]

6.1. THEOREM. *$L_{(i)}$ satisfies ϕ_1 and ϕ_2 for any $a \in Price$ and $d \in Item$:*

$$\phi_1 = [\mathsf{T}^* \cdot \mathbf{send}_{C\to S}(a, h(d), V) \cdot (\neg(\mathbf{send}_{C\to S}(\mathit{no}) \vee \mathbf{send}_{V\to S}(\neg \mathit{ack}(a)) \vee \mathbf{recv}_{C\leftarrow S}(d))))^*] \langle \mathsf{T}^* \cdot \mathbf{recv}_{C\leftarrow S}(d)\rangle \mathsf{T}$$
$$\phi_2 = [\mathsf{T}^* \cdot \mathbf{send}_{C\to S}(a, h(d), V) \cdot (\neg(\mathbf{send}_{C\to S}(\mathit{no}) \vee \mathbf{send}_{V\to S}(\neg \mathit{ack}(a)) \vee \mathbf{recv}_{V\leftarrow B}(\mathit{transfer}(a)))))^*] \langle \mathsf{T}^* \cdot \mathbf{recv}_{V\leftarrow B}(\mathit{transfer}(a))\rangle \mathsf{T}$$

[6] Recall that the lemmas and theorems of this section are all established using finite state model checking.

This theorem, intuitively, states that if no time out happens, in $L_{(i)}$, C will receive the item it desires (ϕ_1) and the bank will transfer the corresponding price to V's account (ϕ_2), under fair scheduling (à la definition 4.1). To complete the picture, we need to ensure that V's money is not revoked afterwards. This is guaranteed by lemmas 6.2 and 6.3.

Timeliness: Timeliness states that each honest process can finalise the exchange with no help from the opponent, cf. § 5.5.3. From C's point of view, finalising the exchange means that it either receives d or $\neg h(d)$ from S, or it deliberately sends *no* to S to cancel the exchange. As a shorthand we write $terminate(C) = \mathbf{recv}_{C \leftarrow S}(d) \vee \mathbf{recv}_{C \leftarrow S}(\neg h(d)) \vee \mathbf{send}_{C \rightarrow S}(\text{no})$. We prove the following lemma for timeliness of C.

6.2. THEOREM. *In $L_{(iii)}$, the following property holds for any $d \in Item$ and $a \in Price$:*

$$[\mathsf{T}^* \cdot \mathbf{send}_{C \rightarrow S}(a, h(d), V) \cdot (\neg terminate(C))^*]\langle(\neg \mathbf{recv}^\dagger(\_))^* \cdot terminate(C)\rangle\mathsf{T}$$

The termination condition for V is more involved. Similar to theorem 6.2, we at least require that in $L_{(ii)}$, V can always reach $\mathbf{recv}_{V \leftarrow B}(transfer(a))$ or $\mathbf{recv}_{V \leftarrow B}(\neg transfer(a))$, without any help from C. This is however not a sufficient condition for V's termination, because the transferred money can in general be revoked. Therefore, in the following formula, we write $terminate^{\frac{1}{2}}(V) = \mathbf{recv}_{V \leftarrow B}(transfer(a)) \vee \mathbf{recv}_{V \leftarrow B}(\neg transfer(a))$.

6.3. THEOREM. *In $L_{(ii)}$, the following property hold for any $a \in Price$:*

$$[\mathsf{T}^* \cdot \mathbf{send}_{V \rightarrow B}(a, V) \cdot (\neg terminate^{\frac{1}{2}}(V))^*]\langle(\neg \mathbf{recv}^\dagger(\_))^* \cdot terminate^{\frac{1}{2}}(V)\rangle\mathsf{T}$$

Even after receiving $transfer(a)$ from B, V cannot be sure that in future $revoke_B(a, V)$ does not happen. This is confirmed by model checking the following property in $L_{(ii)}$:

$$\langle\mathsf{T}^* \cdot \mathbf{recv}_{V \leftarrow B}(transfer(a)) \cdot \mathsf{T}^* \cdot revoke_B(a, V)\rangle\mathsf{T}$$

We thus further require that if $revoke_B(a, V)$ is reachable after $\mathbf{recv}_{V \leftarrow B}(transfer(a))$, then it is reachable without any help from C as well. Although this is is a weak guarantee for V, it does not hold in $L_{(ii)}$, as model checking refutes the following claim.

6.1. CLAIM. *In $L_{(ii)}$, the following property holds for any $a \in Price$:*

$$[\mathsf{T}^* \cdot \mathbf{recv}_{V \leftarrow B}(transfer(a))] \\ (\langle\mathsf{T}^* \cdot revoke_B(a, V)\rangle\mathsf{T} \implies \langle(\neg \mathbf{recv}^\dagger(\_))^* \cdot revoke_B(a, V)\rangle\mathsf{T})$$

Model checking refutes claim 6.1 with the following counterexample:

$$\begin{array}{l}\langle\text{initial state}\rangle\\ \mathbf{recv}^{\dagger}_{S\leftarrow C}(a1, h(d1), V)\\ \mathbf{send}_{S\rightarrow V}(a1, h(d1), V)\\ \mathbf{recv}_{V\leftarrow S}(a1, h(d1), V)\\ \mathbf{send}_{V\rightarrow B}(a1, V)\\ \mathbf{recv}_{B\leftarrow V}(a1, V)\\ \mathbf{send}_{B\rightarrow V}(\mathit{transfer}(a1))\\ \mathbf{recv}_{V\leftarrow B}(\mathit{transfer}(a1))\\ \mathbf{recv}^{\dagger}_{S\leftarrow C}(\text{no})\\ \mathbf{send}_{S\rightarrow B}(\neg a1, V)\\ \mathbf{recv}_{B\leftarrow S}(\neg a1, V)\\ \mathit{revoke}_B(a1, V)\\ \langle\text{goal state}\rangle\end{array}$$

This shows a trace of the protocol which reaches $\mathit{revoke}_B(a1, V)$ only with C's contribution. Therefore, the claim of [VPG01] that "the protocol will terminate for any party which behaves correctly (i.e., according to the protocol)" does not hold, cf. [Vog03]. See below for a partial correction to this flaw.

Fairness (cf. § 2.2): We split up the notion of fairness into fairness for C and V individually. The following lemma addresses fairness for C: If C does not receive the item, then S inevitably revokes the payment, without any help from V.

6.4. THEOREM. *In $L_{(iii)}$ the following property holds for any $d \in \mathit{Item}$ and $a \in \mathit{Price}$:*

$$\begin{array}{l}[\mathsf{T}^* \cdot \mathbf{send}_{C\rightarrow S}(a, h(d), V)\cdot\\ \qquad (\neg \mathit{revoke}_B(a, V))^* \cdot (\mathbf{send}_{S\rightarrow C}(\neg h(d)) \vee \mathbf{send}_{C\rightarrow S}(\mathit{no}))\cdot\\ \qquad\qquad (\neg \mathit{revoke}_B(a, V))^*]\langle(\neg \mathbf{recv}^{\dagger}(\_))^* \cdot \mathit{revoke}_B(a, V)\rangle\mathsf{T}\end{array}$$

Note that theorem 6.2 ensures that either C receives its desired item or one of the actions $\mathbf{send}_{S\rightarrow C}(\neg h(d))$ or $\mathbf{send}_{C\rightarrow S}(\text{no})$ will occur.

Fairness for V is defined correspondingly: If C receives an item, then V receives the corresponding $\mathit{transfer}(a)$. From lemmas 6.1–6.3 we know that if C gets hold of an item, the corresponding payment is never revoked.

6.5. THEOREM. *$L_{(ii)}$ satisfies the following property for any $d \in \mathit{Item}$ and $a \in \mathit{Price}$:*

$$[\mathsf{T}^* \cdot \mathbf{recv}^{\dagger}_{S\leftarrow C}(a, h(d), V) \cdot (\neg \mathbf{recv}_{V\leftarrow B}(\mathit{transfer}(a)))^* \cdot \mathrm{reveal}(d)]\mathsf{F}$$

Discussions

Using finite state model checking we proved lemmas 6.1– 6.3 and theorems 6.1– 6.5, and refuted claim 6.1. Therefore, the protocol provides effectiveness and fairness. Timeliness is however only achieved for the customer.

Various practical aspects of this protocol are discussed in [VPG01, CT04]. In particular, as a remedy to its timeliness flaw, in [CT04] we propose adding a timeout action to S, such that after experiencing a certain delay between sending message 6 and receiving message 7, S would assume that C's answer is *no* and proceed accordingly. This solution, however, hinges on the assumption that C does not disconnect S altogether. This is a tenable assumption when considering, for instance, services provides for cell phones. In such services, S has to be connected to the network to be of any use for C. A malicious C can then prevent V from knowing the result of the exchange, only at the expense of not using its cell phone, cf. rational exchanges in § 2.2.2.

We remark that fair exchange of time sensitive data is in general believed to be hard. According to Asokan "the optimistic approach is inappropriate to exchange time-sensitive items (e.g., current stock quotes). Fair exchange of time-sensitive items over open networks is a difficult problem, regardless of the technique. Even if we use an online third party (instead of the the optimistic approach), the protocol will not be secure against an attacker who can disrupt communication channels long enough" [Aso98]. See also [Jak95, Syv98].

In this case study, the protocol uses resilient confidential authenticated channels. The intruder's power is therefore mostly restricted to scheduling messages over such channels. We have also analysed a fair non-repudiation protocol [CCT05] which explicitly uses cryptographic apparatus to achieve confidentiality and authenticity over channels, instead of simply assuming these features. This study is however omitted here because of its similarity to the presented case studies. The formal verification of this non-repudiation protocol has been tightly coupled with the design phase of the protocol. The analysis has thus been more focused on optimising and fine tuning the protocol, while designing it, rather than testing a fixed design. In particular, we use model checking to minimise the protocol, such that omitting any term from the messages would endanger either the functionality or the security of the protocol. For a complete report on this analysis see [CCT05].

6.3 Analysing a fair DRM scheme

The fair DRM scheme of [NPG+05, TKJ07] aims at providing a secure environment for secure exchange of *content-right* bundles among trusted computing devices.

Trusted devices are tamper-proof hardware that follow their certified software. They use (e.g. render) each *content* exactly as is instructed by its associated *right*. Our study does not address semantics and derivations of rights in DRM systems. This constitutes a whole separate body of research, e.g. see [PW02]. We focus on formal analysis of the transactional properties of the studied DRM scheme.

A challenge in DRM is that the owners of trusted devices are in general untrusted. They may collude to subvert the protocol. They can, in particular, deliberately switch off their own devices (cf. crash failure in § 2.1.). In our analysis we consider two groups of trusted de-

vices, one group owned by the intruder, and one owned by an *irrational* party. The irrational party uses the protocol in an arbitrary manner, it complies with the protocol, but does not actively try to protect its interests. The intruder as usual controls the communication channels. Figure 6.1 shows a scenario with two trusted devices *td*, in which the intruder controls one trusted device and the other one is owned by $owner$.

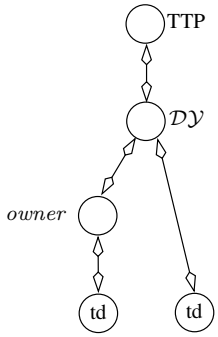


Figure 6.1: *The DRM scheme*

A legitimate content provider, (abusively) referred to as trusted third party (TTP), is also employed in the scheme. The TTP is the original distributor of protected content-right bundles. A content purchased from the TTP can further be traded among the trusted devices, if this is allowed by the associated right. The payment system is assumed also to be managed by the TTP in our analysis, while in practice it can be delegated to a trusted banking server, cf. [NPG+05, TKJ07]. The TTP is assumed impartial in its behaviour and eventually available to respond to requests from trusted devices. The channels between trusted devices and the TTP are assumed to be resilient.

The analysis of this DRM scheme appears in [TKJ07], and largely overlaps with the case study of § 6.2. Modelling trusted devices nonetheless needs certain considerations in our formal analysis. Therefore, in the following, we focus on these issues.

Atomic actions

In the DRM scheme of [NPG+05, TKJ07], trusted devices are able to perform *local atomic* blocks of actions: Multiple actions of each trusted device can be logically linked in the device, such that either all or none of them are executed. Implementing atomic blocks of actions is a well studied topic in concurrency control, e.g. see [MHL+92]. We remark that non-local atomic actions (i.e. atomic actions not necessarily belonging to the same process) can be used to model fair exchange requirements, cf. [Tyg96].

Consider the LTS $L = \partial_H(\mathcal{P} \| I^\dagger(\Gamma_0, \emptyset))$ as defined in § 5. To each action a in L we assign a process name $\aleph(a)$, which reflects the process that performs a, cf. § 7.3.2. The set

of actions that appear in process p are naturally associated with p. Actions resulting from synchronisations, such as a in $a = b|c$, are associated with both $\aleph(b)$ and $\aleph(c)$. For instance, in the model of § 5, if $a = \mathbf{send}_p(m)$, then $\aleph(a) = p$. We say action a *belongs* to process p, iff $\aleph(a) = p$.

6.1. DEFINITION. *We say the set of actions $At = \{a_1, \cdots, a_n\}$ in L is atomic iff in any trace α of L either all or none of a_i, with $i \in \{1, \cdots, n\}$, appear. A set of atomic actions is local iff they belong to the same process and for all α in L, if $\alpha_j = a_i$ and $\alpha_{j'} = a_{i'}$, with $a_i, a_{i'} \in At$, then $\forall j < k < j'.\ \aleph(\alpha_k) = \aleph(a_i) \implies \alpha_k \in At$.*

To model atomic actions in process p we can simply join them using the sequential composition operator $\cdot$. It is clear that either all or none of such actions would appear in any execution of the system (assuming maximal progress, cf. § 5.4.2).

Let us assume a trusted device owned by the intruder be specified by p_t, a process à la § 4.1. To enable the intruder to turn off the device (i.e. p_t), we compute $p_t^{+\mathit{off}}$ by substituting each action a appearing in p with the process $a + \mathit{off}.\delta$. The resulting model of p_t can obviously perform *off* at any moment and cease to act afterwards, thus reflecting the intruder's ability to turn the device off at will. [7]

We note that $p_t^{+\mathit{off}}$ is not able to perform any atomic actions. As mentioned above, atomic actions are modelled as a collection of sequentially composed actions. Let $\{a, b\}$ be a set of atomic actions modelled in p_t as $a \cdot b$. To retain local atomicity, we add the $\mathit{off}.\delta$ alternative to either both or none of them. For $p_t = a \cdot b \cdot p'$ we thus get $p_t^{+\mathit{off}} = ((a \cdot b) + \mathit{off}.\delta) \cdot p'$. This mechanism allows us to model the effect of atomic actions and also the ability of the intruder to turn off a trusted devices at any moment, except when performing atomic blocks of actions, in our formalisation [TKJ07].

Modelling the TTP

In the protocols of [NPG+05, TKJ07] the TTP needs to authenticate the trusted devices before exchanging information with them. This is necessary in order to prevent various replay attacks and right masquerading (see also § 7.5.1). The TTP process therefore requires fresh nonces in each exchange. If we model the TTP as a cyclic process as in § 6.2, then the state space would not be finite, since the TTP can keep on generating fresh nonces. However, limiting the TTP to a finite number of exchanges makes it susceptible to *denial of service* attacks. Namely, the intruder can simply exhaust the TTP process before the honest parties get the chance to resolve their exchanges at the TTP. Note that since the attacker is a legitimate user of the protocol in FE protocols, the TTP cannot simply avoid responding to the attacker's resolve requests. In practice, such attacks are mitigated by putting time limits on transactions with the TTP, recovering the service of the TTP after a short delay, etc.

[7] This construction may not optimised. For instance, for $p = a \cdot p' + b \cdot q$, with $a, b \in Act$, we get $p^{+\mathit{off}} = (a + \mathit{off} \cdot \delta) \cdot p' + (b + \mathit{off} \cdot \delta) \cdot q$, which can be simplified to $p^{+\mathit{off}} = a \cdot p' + b \cdot q + \mathit{off}.\delta$.

As we abstract away from such low-level aspects in our model, instead, the action η is used to indicate that all TTP processes in the model are exhausted by the intruder. In other words, as long as this action has not occurred yet, there is still at least one TTP thread available to resort to. The eventual fair reachability under the $\mathcal{F}^{\dagger}$ constraint is replaced with the following property in this model (recall the property of § 5.5.3).

$$\varphi = [(\neg\eta)^* \cdot \mathtt{1} \cdot (\neg(\bar{\mathtt{1}} \vee \eta))^*]\langle(\neg\mathbf{recv}^{\dagger}(\_))^* \cdot \bar{\mathtt{1}}\rangle\mathsf{T}$$

This property intuitively states that if 1 happens, while the TTP resolves disputes, there is a fair path (with no help from the intruder) to $\bar{\mathtt{1}}$. The simplicity of this solution shows the flexibility of our protocol and property specification formalisms, when addressing protocol specific constraints.

Chapter 7

Partial order reduction for security protocols

7.1 Introduction

Two main approaches to automatic verification of security protocols are model checking and constraint solving. Both these techniques in principle need to enumerate all possible interleavings of actions performed by protocol participants. Partial order reduction (POR) techniques identify and avoid generating identical interleavings, modulo the properties that are to be verified, to reduce the time and memory used in verification. Clarke, Jha and Marrero [CJM00a, CJM03] were the first to formally present a POR algorithm for security protocols and determine the class of modal properties that are preserved by it. They observe that the knowledge of the Dolev-Yao intruder model in the course of each protocol run is non-decreasing and, intuitively, with more knowledge the intruder can do more (harm). Therefore, when verifying security protocols which yield finite executions, in the presence of the $\mathcal{DY}$ intruder, it is safe to prioritise actions that increase the intruder's knowledge over other actions. This is the heart of the POR algorithm of [CJM00a], which was originally used in BRUTUS [CJM00b], a model checker tailored for security protocols. This algorithm has also been mentioned in [MS01] as a means to reduce the number of constraint sets that have to be solved in order to verify a security protocol.

The POR algorithm of [CJM00a] assumes that security protocols are *non-branching*, meaning that each participant at each state of the protocol has at most one single action to perform. [1] This assumption has been widely used in the literature for modelling various authentication and key distribution schemes since the early years of security protocol analysis, e.g. see the ping-pong protocols of [DY81].

In practice, however, participants of, e.g., authentication protocols may have *choice points*. They can for instance take alternative actions when a received message does not match a certain pattern, or when a timeout occurs. Therefore, any faithful model of these protocols has to allow such choice points in the specification as well. More importantly, some security protocols inherently prescribe more than one possible behaviour for the participants. Prominent examples of such security protocols are optimistic fair exchange protocols, including non-repudiation, fair payment, certified email and electronic contract signing protocols

[1] Note that the notion of non-branching refers to the participating processes, although it is used as a qualifier for protocols. Naturally, non-branching protocols are not necessarily deterministic, see § 7.2.1.

(see § 2.2). Participants of these protocols can choose between continuing the normal flow of the protocol and resorting to a trusted entity, in case of long delays of the opponent or communication failures. These protocols can be properly modelled only if choice points are allowed in the specification of their participants.

Road map In this chapter, we extend the POR algorithm of Clarke et al. to handle branching security protocols, i.e. protocols in which participants have choice points. To achieve this, in § 7.2 we present an enriched model of security protocols that explicitly allows conditionals and choice points in the specification of participants. § 7.3 motivates why the algorithm of [CJM00a] falls short in addressing branching security protocols. There, we also present our extension to this algorithm.

Our focus is on how the extended POR algorithm can be used in explicit state model checking of security protocols. The application of the extended POR algorithm to constraint solving for security protocols is briefly discussed in § 7.4. Some experimental results are presented in § 7.5. In § 7.6, we conclude this chapter with comparing our work to existing POR techniques for security protocols and discussing some future research directions.

7.2 Preliminaries

The model we use in this chapter is in some ways more restrictive, compared to the model of § 5. We thus prefer to define some basic concepts anew. In particular, here, we do not allow *direct* communication between honest processes, while this is allowed in § 5. This condition is necessary to ensure that each action of the protocol changes the state of at most one honest process (this is used in lemma 7.1). More importantly, cyclic participants are excluded from our current model, see our future work § 7.6.

7.2.1 Modelling security protocols

We model a security protocol $\mathcal{P}$ as a finite number of honest message-passing processes $\{p_1, \cdots, p_n\}$ which communicate through the $\mathcal{DY}$ intruder. The intruder comprises all corrupted parties and, also, controls the communication media. We start with defining the set of messages, cf. § 4.1.2.

7.1. DEFINITION. *Let MF be a set of function symbols and MV be a set of message variables. The set of messages Msg is inductively defined as:*

- $m \in MV \implies m \in Msg$
- $m_1, \cdots, m_k \in Msg \implies f(m_1, \cdots, m_k) \in Msg$, *if f is a k-ary function symbol in MF. For function symbols of arity* 0 *(serving as constants), instead of $f()$ we write f.*

The function $var : Msg \to 2^{MV}$ returns all variables which occur in a message. We define $var(M) = \cup_{m \in M} var(m)$. A messages m is called *closed* iff $var(m) = \emptyset$. The set of all

closed messages is Msg^c. Let σ be a partial function $\sigma : MV \rightarrow Msg^c$. The domain of σ is denoted by $d(\sigma)$. For a message $m \in Msg$, we obtain $m\sigma$ by simultaneously substituting all variables $v \in var(m) \cap d(\sigma)$ in m with their corresponding $\sigma(v)$.

Below, we describe the behaviour of honest processes. Each honest process is an (extended) LTS (Σ, s_0, A, Tr), with the extra condition that each action $a \in A$ is a pair $a = (l, m)$, also denoted $l(m)$, where l is the action's *label* and $m \in Msg$ is the action's *parameter*. When $a = l(m)$, we write $\lambda(a) = l$ and $\Omega(a) = m$. The set of all action labels of A is defined as $\lambda(A) = \cup_{a \in A} \lambda(a)$. We say that A is *closed* iff $var(\Omega(A)) = \emptyset$, where $\Omega(A) = \cup_{a \in A} \Omega(a)$.

LTSs describing honest processes are required to be finite, acyclic, single-image. A protocol $\mathcal{P} = \{p_1, \cdots, p_n\}$ is called *non-branching* iff all p_i are deterministic. Otherwise, $\mathcal{P}$ is called *branching*. We remark that both branching and non-branching protocols in general result in non-deterministic LTSs (see the definitions of § 4.1.1).

To interact with the communication media, a process $p \in \mathcal{P}$ has two designated actions $send_p(\langle q, m \rangle)$ and $recv_p(\langle p, m \rangle)$, in which message $m \in Msg$ is produced and consumed, respectively, while q is the identity of the intended recipient process (so that m can be routed to its destination). We may omit the address tag q from $\langle q, m \rangle$ when confusion is unlikely. Apart from $send$ and $recv$, all actions of honest processes are assumed *internal*, i.e. not communicating with other processes.

Internal actions typically denote security claims of protocol participants or their internal decisions. We assume that all internal actions of process p contain p as a subscript. For example, $\mathsf{secret}_p(m)$ can be an internal action performed by p when concluding that m is a secret. Note that internal actions can also have messages as parameters.

Let $p \in \mathcal{P}$ be an honest process described by $L_p = (\Sigma_p, s_{0p}, A_p, Tr_p)$. We partition $\lambda(A_p)$ into three mutually disjoint sets: R_p, S_p and I_p, where $R_p = \{recv_p\}$, $S_p = \{send_p\}$ are $I_p = \{\lambda(a_p) \mid a_p \text{ is an internal action of } p\}$. Since all the actions of processes are subscripted with their corresponding process names, we have $A_p \cap A_q = \emptyset$, for any two different processes $p, q \in \mathcal{P}$. To avoid name clashes, we further assume $var(\Omega_{A_p}) \cap var(\Omega_{A_q}) = \emptyset$.

Each process $p \in \mathcal{P}$ is enriched with a special set of internal action labels $B_p \subseteq I_p$ that model its internal choices. These action labels can behave as Boolean functions and affect the execution flow of the process, i.e. p may use the results of these functions to decide which branch to follow (cf. the *check* primitive in [CE02]). For each action label in B_p we thus have

$$\forall l_p \in B_p.\ l_p : Msg^c \rightarrow \{\mathsf{T}, \mathsf{F}\} \tag{7.1}$$

Below we define *well-formed* processes. Intuitively, a well-formed process can only send, and decide based on, closed messages in the course of any protocol execution. However, $recv$ actions may contain, and subsequently instantiate, non-closed parameters. [2]

[2] Our model of protocol participants is in many ways similar to the strand space formalism of [JHG99].

7.2. DEFINITION. *A process p is called well-formed iff the following property holds in any trace α in L_p:*

$$\forall i.\ \lambda(\alpha_i) \notin R_p \implies var(\Omega(\alpha_i)) \subseteq \cup_{1 \leq j < i} var(\Omega(\alpha_j)).$$

We require all members of $\mathcal{P}$ (i.e. honest participants) to be well-formed processes.

Now, we turn to modelling the $\mathcal{DY}$ intruder. We do not explicitly model $\mathcal{DY}$ as a separate process. The semantics of security protocols, described below, implicitly models the behaviour of the intruder. As always, $\mathcal{DY}$ comprises all malicious participants and, moreover, controls the entire communication network (cf. § 5). The latter feature implies that all messages are channelled through the intruder and the intruder is always ready to receive messages from other processes. We follow the convention that $\mathcal{DY}$ adds all the intercepted messages to its *knowledge*, which is a set of messages. We assume that the state of the intruder is uniquely described with its knowledge set. The $\mathcal{DY}$ intruder can also send a closed message, if it can deduce the message from its knowledge. We do not explicitly define the deduction function $\cdot \vdash \cdot : Msg^c \times 2^{Msg^c} \to \{\mathsf{T}, \mathsf{F}\}$, as our results do not depend on its features, except for its *monotonicity*: $\forall m \in Msg^c, X, Y \in 2^{Msg^c}.\ X \subseteq Y \implies (X \vdash m \implies Y \vdash m)$. Note that the deduction system of definition 2.2 is indeed monotonic.

In our model, $\mathcal{DY}$ is not allowed to perform internal actions by itself. This is because intruder internal actions in principle do not change the knowledge of the intruder, while they may change the state of the intruder. This would contradict the aforementioned assumption that the state of the intruder is an injective function of its knowledge. Intruder's internal actions may however be useful in referring to $\mathcal{DY}$'s state, as in, e.g., § 7.5.1.

To circumvent this shortcoming, we can add a dummy process dy to $\mathcal{P}$, which is triggered only by $\mathcal{DY}$. When $\mathcal{DY}$ wishes to perform an "internal" action $a_{\mathcal{DY}}(m)$, it would instruct dy, e.g. by sending a designated message, to perform the corresponding action $a_{dy}(m)$ (which is internal to dy). Otherwise, dy is modelled as an honest process. For convenience, below, we subscript the internal actions of dy with $\mathcal{DY}$, i.e. we write $a_{\mathcal{DY}}$ for a_{dy}.

Semantics

We attribute an asynchronous message exchange semantics to security protocols, as is defined below. Let $p_1 = (\Sigma_1, s_{0_1}, A_1, Tr_1), \cdots, p_n = (\Sigma_n, s_{0_n}, A_n, Tr_n)$ be n honest participants of protocol $\mathcal{P} = \{p_1, \cdots, p_n\}$, which runs in the presence of the $\mathcal{DY}$ intruder. We let Γ_0 denote the initial knowledge of $\mathcal{DY}$.

7.3. DEFINITION. *The* product *of processes $p_1, \cdots, p_n$ and $\mathcal{DY}$, denoted $p_1 \otimes \cdots \otimes p_n \otimes \mathcal{DY}$, is an LTS $L = (\Sigma, s_0, A, Tr)$, in which each state $s = \langle x_0, \cdots, x_n, \Gamma, \sigma \rangle$ contains the state of each process of $\mathcal{P}$, the intruder knowledge Γ and also a partial function $\sigma : MV \to Msg^c$. The sets Σ, Tr and A are the smallest sets which satisfy the following conditions:*

- *Initial state:* $s_0 = \langle s_{0_1}, \cdots, s_{0_n}, \Gamma_0, \emptyset \rangle \in \Sigma$.

- *Internal actions:*
  If $\langle x_0, \cdots, x_i, \cdots, x_n, \Gamma, \sigma\rangle \in \Sigma$, *and* $\exists m' \in Msg,\ l_{p_i} \in I_{p_i}.\ (x_i, l_{p_i}(m'), x_i') \in Tr_i$, *then* $\langle x_0, \cdots, x_i, \cdots, x_n, \Gamma, \sigma\rangle \xrightarrow{l_{p_i}(m)} \langle x_0, \cdots, x_i', \cdots, x_n, \Gamma, \sigma\rangle \in Tr$ *with* $m = m'\sigma$. *If* $l_{p_i} \in B_{p_i}$ *(see equation 7.1), then it is additionally required that* $l_{p_i}(m) = \mathsf{T}$. *Note that* $m \in Msg^c$, *since* p_i *is well-formed.*

- $\mathbf{send}$ *actions:*
  If $\langle x_0, \cdots, x_i, \cdots, x_n, \Gamma, \sigma\rangle \in \Sigma$ *and* $\exists m' \in Msg.\ (x_i, send_{p_i}(m'), x_i') \in Tr_i$, *then the transition* $\langle x_0, \cdots, x_i, \cdots, x_n, \Gamma, \sigma\rangle \xrightarrow{\mathbf{send}_{p_i}(m)} \langle x_0, \cdots, x_i', \cdots, x_n, \Gamma \cup \{m\}, \sigma\rangle$ *belongs to* Tr, *with* $m = m'\sigma$. *Note that* $m \in Msg^c$, *because* p_i *is well-formed.*

- $\mathbf{recv}$ *actions:*
  If $\langle x_0, \cdots, x_i, \cdots, x_n, \Gamma, \sigma\rangle \in \Sigma$ *and* $\exists m' \in Msg.\ (x_i, recv_{p_i}(m'), x_i') \in Tr_i$, *then the transition* $\langle x_0, \cdots, x_i, \cdots, x_n, \Gamma, \sigma\rangle \xrightarrow{\mathbf{recv}_{p_i}(m)} \langle x_0, \cdots, x_i', \cdots, x_n, \Gamma, \sigma'\rangle$ *belongs to* Tr *if* $m = m'\sigma' \in Msg^c \wedge\ \Gamma \vdash m$. *Moreover,* σ' *should satisfy* $\sigma \subseteq \sigma'$ *and* $d(\sigma') = d(\sigma) \cup var(m')$.

We fix the notation and write L_f (standing for the *full* LTS), described by $(\Sigma_f, s_{0f}, A_f, Tr_f)$, for the product $p_1 \otimes \cdots \otimes p_n \otimes \mathcal{DY}$. We define the function $K : \Sigma_f \to 2^{Msg^c}$ to return the intruder's knowledge set at a given state.

Observe that the set of labels of elements of A_f can be partitioned into three disjoint sets: $\lambda(A_f) = \mathbf{R} \cup \mathbf{S} \cup \mathbf{I}$, where $\mathbf{R} = \cup_{p\in\mathcal{P}}\{\mathbf{recv}_p\}$, $\mathbf{S} = \cup_{p\in\mathcal{P}}\{\mathbf{send}_p\}$ and $\mathbf{I} = \cup_{p\in\mathcal{P}} I_p$. We choose to write the elements of $\mathbf{S}$ and $\mathbf{R}$ in bold face, i.e. $\mathbf{send}$ and $\mathbf{recv}$, in contrast to $send$ and $recv$. The action $\mathbf{recv}_p$ is the result of the communication between the intruder process (which we do not explicitly specify) and p performing a $recv_p$ action (and similarly for $\mathbf{send}_p$ and $send_p$).

We remark that performing a $\mathbf{recv}_p(m)$ action depends not only on the state of p, but also on the intruder's ability to construct the message m. Whereas for a $\mathbf{send}_p(m)$ to happen, no condition is put on the intruder's state. Similarly, internal actions of a process can happen with no conditions on other processes' or the intruder's states.

It is worth mentioning that L_f is single-image. This is because all processes p_i are single-image, $A_{p_i} \cap A_{p_j} = \emptyset$ for $i \neq j$ and, moreover, the communication actions ($\mathbf{send}_p(m)$ and $\mathbf{recv}_p(m)$) are uniquely named. [3] As we only consider well-formed processes, A_f is closed. Moreover, L_f is acyclic since all p_i are (by definition) acyclic. However, we cannot claim that L_f is finite based on the finiteness of p_i, because the $\mathcal{DY}$ intruder is not necessarily finite. In fact, it can compose an infinite number of messages with consecutively pairing a single constant value (see § 2.1.1). However, usually model checking algorithms hinge on the finiteness of the model, and so does our proposed POR algorithm. Therefore, in the

[3] A single $recv_p(\langle p, v\rangle)$ action in p, with $v \in MV$, can produce several $\mathbf{recv}_p$ actions in L_f. These will however have distinct closed messages assigned to v.

following discussions we assume that L_f is finite (for a complementary discussion, see § 7.4 on POR for constraint solving approaches). In current practice of model checking security protocols, finiteness of the model can be achieved by, for instance, assuming typed messages (see § 4.1.2).

7.2.2 State space generation

A state space generation algorithm for security protocols is provided with the LTSs $p_1, \cdots, p_n$ as input specification, and generates L_f (see definition 7.3), as output.

Algorithm 7.1 shows a typical generation algorithm for finite LTSs (in our model, security protocols are assumed to result in finite LTSs, see § 7.2.1). Here, we confine to the traversal strategy and abstract away from generating the output (file). In algorithm 7.1, $Open$ is the set of visited, but not yet expanded states, and $Closed$ is the set of visited and expanded states. When $Open$ is implemented as a queue, the resulting traversal strategy is *breadth-first*, while implementing $Open$ as a stack results in a *depth-first* strategy. Definition 7.2.1 determines s_0 and the value of en for each state, in the given specification (recall the notations of § 4.1.1).

```
1: Open := ∅; Closed := ∅
2: Open.insert(s_0)
3: while Open ≠ ∅ do
4:    s := Open.extract
5:    Closed.insert(s)
6:    for all a ∈ en(s) do
7:       if a(s) ∉ Closed ∧ a(s) ∉ Open then
8:          Open.insert(a(s))
```

Algorithm 7.1: *State space generation*

7.2.3 Partial order reduction

The main principle of POR is to exploit the commutativity of concurrently executed actions in order to generate only a sufficient fraction of the state space. For general introductions to POR see [PPH97, CGP00]. Here, we mainly follow the *ample set* method [Pel97, Pel98].

A POR algorithm prescribes a method to select $ample(s)$ at each state s, with $ample(s) \subseteq en(s)$, such that exploring only elements of $ample(s)$, instead of the entire $en(s)$, is enough to *preserve* a certain class of desired properties. This corresponds to changing line 6 of algorithm 7.1 to **for all** $a \in ample(s)$ **do**.

For a given specification, let L and L' be the full LTS generated by an exhaustive generation algorithm (e.g. algorithm 7.1) and the LTS produced by a POR algorithm por, respectively. For a set of trace properties Φ, we say por preserves Φ iff $\forall \phi \in \Phi.\ L \models \phi \iff L' \models$

ϕ. We aim at the trace properties that are expressible in $\mathbf{LTL}_{-\mathbf{X}}$, as the class of properties preserved by our POR algorithm. Below, we describe the syntax of $\mathbf{LTL}_{-\mathbf{X}}$ properties.

7.4. DEFINITION. *Given a set of actions AP, which at least contains the symbolic action $\top$, the set of formulae of* $\mathbf{LTL}$ *is inductively defined as:* [4]

- *Every member of AP belongs to* $\mathbf{LTL}$.
- *If ϕ_1 and ϕ_2 belong to* $\mathbf{LTL}$*, then so do $\neg\phi_1$, $\phi_1 \vee \phi_2$, $\mathcal{X}\phi_1$ and $\phi_1 \mathcal{U} \phi_2$.*

The connectives $\wedge$ and $\implies$ are defined in the standard way. The formula $\phi_1 \mathcal{U} \phi_2$, intuitively, expresses that eventually ϕ_2 will become true and, before that, property ϕ_1 holds all the time. As a convention, we write $\Diamond\phi$ for $\top \mathcal{U} \phi$ and $\Box\phi$ for $\neg\Diamond(\neg\phi)$.

The semantics of $\mathbf{LTL}_{-\mathbf{X}}$ formulae is given below. We write $L, s \models \phi$ if ϕ, a formula expressed in $\mathbf{LTL}$, holds in all traces $\alpha \in \pi(s)$ in LTS L. When $L, s_0 \models \phi$, we may write $L \models \phi$. We simply write $s \models \phi$ when L is clear from the context. The relation $\models$ is inductively defined below. For trace $\alpha = \alpha_1 \cdots \alpha_i \cdots$, we define $\alpha^{i\infty} = \alpha_i \cdots$, when i is less than or equal to the length of α.

- $L, \alpha \models a$, with $a \in AP$, iff $\alpha_1 = a$ or $a = \top$.
- $L, \alpha \models \neg\phi$ iff $\neg(L, \alpha \models \phi)$.
- $L, \alpha \models \phi_1 \vee \phi_2$ iff $L, \alpha \models \phi$ or $L, \alpha \models \phi_2$.
- $L, \alpha \models \mathcal{X}\phi$ iff $L, \alpha^{2\infty} \models \phi$.
- $L, \alpha \models \phi_1 \mathcal{U} \phi_2$ iff $\exists k.\ L, \alpha^{k\infty} \models \phi_2 \wedge \forall j. 1 \leq j < k \implies L, \alpha^{j\infty} \models \phi_1$. [5]

The logic $\mathbf{LTL}_{-\mathbf{X}}$ is defined by removing the next time operator $\mathcal{X}$ from $\mathbf{LTL}$. This operator generally defies effective POR [Pel97].

We continue with some preliminary definitions. Consider an arbitrary LTS L. For each trace $\alpha = \alpha_1 \cdot \alpha_2 \cdots$ in L, we label the state s_i^α with α_{i+1} (recall the definitions of § 4.1.1). The label assigned to state s is denoted by $label(s)$. An action a is called *invisible* with respect to a set $AP' \subseteq AP$ iff for each s and s' such that $s' = a(s)$, we have $label(s) \cap AP' = label(s') \cap AP'$. Otherwise, a is called *visible*. If $a \notin AP$, then clearly a is an invisible action for the class of properties expressible in $\mathbf{LTL}_{-\mathbf{X}}$. An *independence* relation $IND \subseteq A \times A$ is the largest anti-reflexive relation, such that for each state s and each $(a, a') \in IND$, if $s_1 = a(s)$ and $s_1' = a'(s)$, then $\exists s_2.\ s_2 = a'(s_1) \wedge\ s_2 = a(s_1')$. The *dependence* relation DEP is defined as $DEP = A \times A \setminus IND$. Two actions a and a' are called *dependent* iff $(a, a') \in DEP$. Now we recall a well-known theorem on $\mathbf{LTL}_{-\mathbf{X}}$-preserving POR in general LTSs (e.g. see [Pel97]).

[4] We note that $\mathbf{LTL}$ formulae are originally defined on Kripke structures, where AP represents a set of atomic propositions attributed to states. We consider an action based variant of $\mathbf{LTL}$, as our models are LTSs, rather than Kripke structures. Such translations have been well studied in the literature [NV95, Pel97, Gia99].

[5] For $\mathcal{U}$ and $\mathcal{X}$ to be well-defined, the transition relation of L has to be total.

7.1. THEOREM. *If the $ample$ set of a POR algorithm satisfies the following properties, then the POR algorithm preserves* $\mathbf{LTL}_{-\mathbf{X}}$.

C0. $ample(s) \subseteq en(s)$*; and* $ample(s) = \emptyset$ *only if* $en(s) = \emptyset$.

C1. Along every trace in the full state space that starts at s, the following condition holds: An action (outside $ample(s)$) that is dependent on an action in $ample(s)$ cannot be executed without an action in $ample(s)$ occurring first.

C2. If $ample(s) \neq en(s)$, then every $a \in ample(s)$ is invisible.

C3. No cycle contains a state in which some action a is enabled, but is never included in $ample(s)$ for any state s on the cycle.

Intuitively, C0 is a sanity check. Note that C1–C3 do not prevent selecting $\forall s.\ ample(s) = \emptyset$, while this trivial case is excluded by C0. Condition C1, roughly speaking, states that in any trace $\alpha \in \pi(s)$, in the full LTS, if $\alpha_1 \notin ample(s)$, then the actions of α can be permutated such that the resulting trace starts with an action in $ample(s)$ and, moreover, belongs to $\pi(s)$. See also [Pel97, Pel98] for intuitive explanations of C1. Below, via an example, we informally describe the idea behind the conditions C2 and C3.

7.1. EXAMPLE. *Figure 7.1 shows L_1, in which actions a_p and b_q are independent. We are interested in property $\phi = \Diamond a_p$. Clearly, $L_1 \models \phi$.*

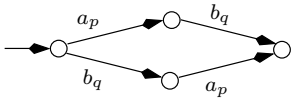


Figure 7.1: *LTS L_1 of example 7.1.*

Intuitively, since a_p and b_q are independent, if only one of them is explored at the initial state, the resulting LTS also satisfies ϕ. We can thus assign $ample(s_0) = b_q$.

Now, consider L_2 of figure 7.2, in which a_p and b_q are indeed independent. Clearly, exploring only b_q at s_0 results in an LTS in which ϕ does not hold, although $L_2 \models \phi$. Condition C3 prevents this choice for $ample(s_0)$ in L_2.

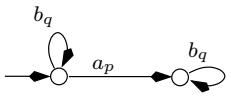


Figure 7.2: *LTS L_2 of example 7.1.*

This $ample$ set is also not suitable in L_1 when we are interested in property $\phi' = \Diamond(b_q \wedge \Diamond a_p)$. In this case, C2 prevents us from exploring only b_q at s_0, since b_q is visible in ϕ'.

We use theorem 7.1 to prove that our proposed POR algorithm preserves $\mathbf{LTL}_{-\mathbf{X}}$ properties. Regarding L_f, we assume that $\lambda(AP) \subseteq \mathbf{R} \cup \mathbf{I}$, i.e. only $\mathbf{recv}$ actions and internal actions can explicitly be referred to in the properties. The set of visible internal action labels is denoted by $\mathbf{V}$, i.e. $\mathbf{V} = \mathbf{I} \cap \lambda(AP)$.

7.3 POR for security protocols

In this section, first, we briefly describe the POR algorithm of [CJM00a]. This is a POR algorithm tailored for verifying security-related properties of non-branching protocols, such as authentication and key distribution protocols. Next, we extend this algorithm to cover branching security protocols, such as optimistic FE protocols.

7.3.1 POR for non-branching security protocols

In [CJM00a] it is observed that the knowledge of the $\mathcal{DY}$ intruder is non-decreasing, and with more knowledge more states are reachable to the intruder. This intuitively implies that $\mathbf{send}$ actions, which typically increase the intruder's knowledge, can be prioritised over other actions in security protocols. This is the heart of the POR algorithm for non-branching security protocols that is proposed in [CJM00a]. The set of actions to be explored at each state s, namely $ample(s)$, is chosen in [CJM00a] as follows:

- If $en(s)$ contains an invisible internal action, then $ample(s)$ is a singleton containing an arbitrary invisible action picked from $en(s)$.
- Suppose $en(s)$ does not contain an invisible internal action, but does contain a $\mathbf{send}$ action. Then, $ample(s)$ is a singleton containing an arbitrary $\mathbf{send}$ action of $en(s)$.
- If $en(s)$ does not contain an invisible action or a $\mathbf{send}$ action, then $ample(s) = en(s)$.

Via an example we give an informal reason why $\mathbf{recv}$ actions cannot be prioritised over $\mathbf{send}$ actions.

7.2. EXAMPLE. *Figure 7.3 depicts processes p and q and the product $p \otimes q \otimes \mathcal{DY}$, with $\Gamma_0 = \{t_0\}$. There, it is assumed that $f \in MF$ and $v \in MV$ may only be instantiated with closed messages of a certain type* `t`*. The only members of* `t` *are t_0 and t_1. We are interested in checking the $\mathbf{LTL}_{-\mathbf{X}}$ property $\phi = \Diamond \mathbf{send}_p(f(t_1))$ in this system. Note that if the POR algorithm prioritises $\mathbf{recv}_p(t_0)$ action over $\mathbf{send}_q(t_1)$, then it would not preserve ϕ. Similarly, none of the $\mathbf{recv}_p(t_0)$ and $\mathbf{recv}_p(t_1)$ may be prioritised over the other one.*

Here, a key observation is that possibly $K(s) \subset K(s')$, when s' is reached from s by a $\mathbf{send}$ action. Since $\vdash$ is monotonic, generally more $\mathbf{recv}$ actions can be instantiated at s' than s. Note that since the participating processes are assumed to be acyclic, no infinite sequence

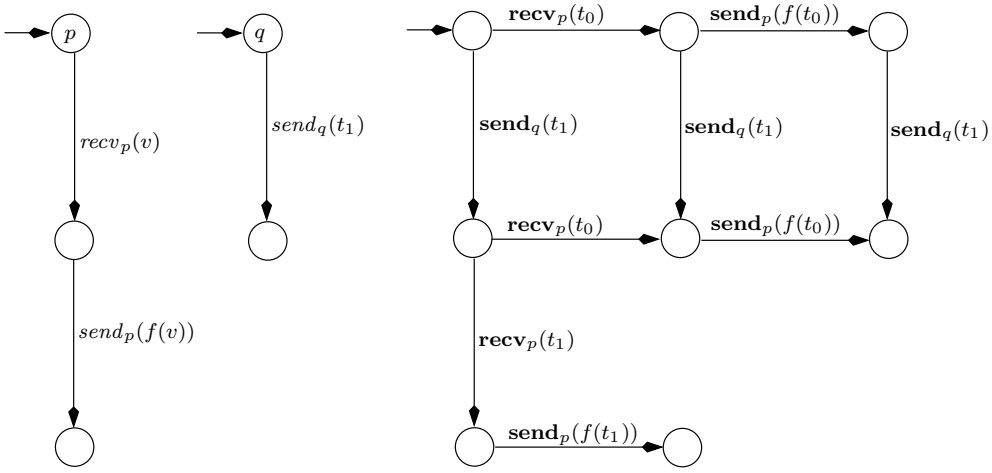


Figure 7.3: *LTSs of example 7.2.*

of **send** actions is possible. This is crucial for the correctness of this algorithm, because, otherwise, the algorithm would run into such a sequence and indefinitely ignore exploring other possibilities.

To specify the requirements of security protocols, Clarke et al. consider a first order logic with a past time modal operator, where quantifiers range over finite sets of protocol participants, etc. (quantifiers thus serve merely as syntactic shorthands). They augment this logic with explicit epistemic operators. Their POR algorithm is shown to preserve formulae of this logic. For a comparison of the logic of [CJM00a] with $\mathbf{LTL}_{-\mathbf{X}}$, see § 7.6.

7.3.2 POR for branching security protocols

Motivations

The POR algorithm of [CJM00a] is not suitable for branching security protocols. Examples of branching security protocols are various optimistic fair exchange protocols, see § 2.2. Participants of these protocols can usually choose between multiple $send$, $recv$ and internal actions. These are therefore modelled as finite, acyclic, single-image, but non-deterministic, processes. Taking only one **send** from a process into the $ample$ set is not safe in these cases, because it can in principle disable other actions of that process. The following example clarifies the idea.

7.3. EXAMPLE. *Figure 7.4 shows processes p and q, along with the LTS of $p \otimes q \otimes \mathcal{DY}$, where $\Gamma_0 = \emptyset$. Note that the dotted transitions and states will never be explored, if the POR algorithm of [CJM00a] is used. In this case, undesirably, it is not detected that the intruder can in fact learn t', hence missing a potential security breach.*

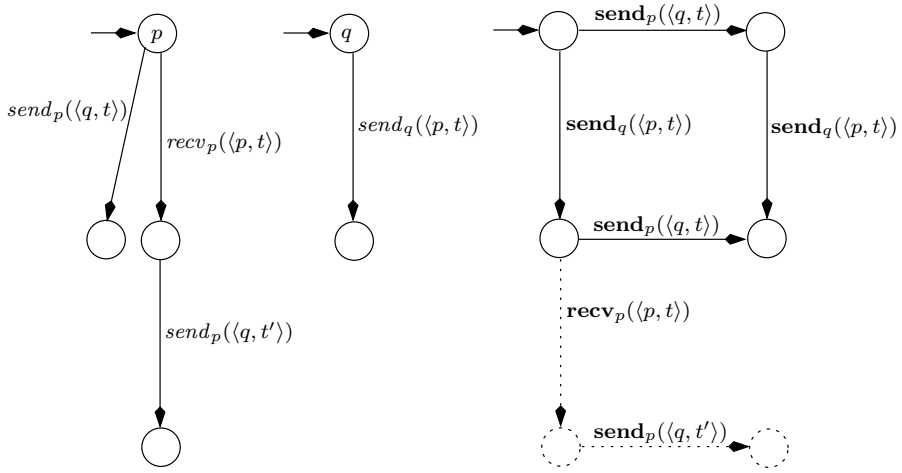


Figure 7.4: *The LTSs of example 7.3.*

This motivates extending the POR algorithm of [CJM00a]. Below, we introduce our POR algorithm for branching security protocols. Next, we show that the proposed POR algorithm indeed preserves $\mathbf{LTL}_{-\mathbf{X}}$ properties.

POR algorithm

Recall that $p_1 \otimes \cdots \otimes p_n \otimes \mathcal{DY}$ is described by the LTS $L_f = (\Sigma_f, s_{0f}, A_f, Tr_f)$. We follow the idea of [CJM00a]: We prioritise transitions with action labels of the set $\mathbf{V}^c \cup \mathbf{S}$ over those of $\mathbf{R} \cup \mathbf{V}$, where $\mathbf{V}^c = \mathbf{I} \setminus \mathbf{V}$. However, since processes can branch in our model, if one action of process p is taken into the $ample$ set, then all other actions of p enabled at that state should be taken as well. Hence, actions of $\mathbf{V}^c \cup \mathbf{S}$ can be prioritised over others only if their corresponding process does not perform any action from $\mathbf{V} \cup \mathbf{R}$ at that state. This raises a new problem, which does not appear in non-branching protocols. The problem is described below.

Let p be a process and, consider the following two scenarios: First, the case where p can only perform one $send_p$ action at state $s \in \Sigma_p$. Second, the case where p can perform $send_p$ and $recv_p$ at state s, but the $recv_p$ action of p is not present in L_f as $\mathbf{recv}_p$, because the intruder does not have enough knowledge to compose that message (cf. definition 7.3). These two scenarios are represented in exactly the same way in L_f, [6] while the POR algorithm has to distinguish them. This is because in the first scenario POR can prioritise p's action over other actions, while in the second scenario it cannot.

[6] The problem that is discussed here is specific to explicit state space generation and is in principle not relevant to constraint solving approaches, cf. § 7.4.

A solution to this problem is to statically pre-process the LTSs of the participating processes by adding a dummy κ_p action as an always-available alternative to $recv_p$ actions in every LTS p. We assume that κ_p actions are solely used for this purpose in the specification.

7.5. DEFINITION. *Given a process $p = (\Sigma_p, s_{0_p}, A_p, Tr_p)$, a κ-translation of p is an LTS $p^\kappa = (\Sigma_p, s_{0_p}, A_p \cup \{\kappa_p\}, add_\kappa(Tr_p))$, where add_κ is defined as $add_\kappa(\emptyset) = \emptyset$ and*

$$add_\kappa(\{(s,a,s')\} \cup T) = \begin{cases} \{(s,\kappa_p,s')\} \cup \{(s,a,s')\} \cup add_\kappa(T) & \text{if } \lambda(a) = recv_p \\ \{(s,a,s')\} \cup add_\kappa(T) & \text{otherwise} \end{cases}$$

The internal actions κ_p can subsequently be used to detect the existence of non-enabled $recv_p$ actions at each state, even when the corresponding $\mathbf{recv}_p$ action is not present in L_f. This intermediate translation step is also required to relate the reduced LTS to L_f, as is described below.

Given a specification of a set of honest processes $p_1, \cdots, p_n$, our POR algorithm is performed in two steps. We first construct $p_1^\kappa, \cdots, p_n^\kappa$, following definition 7.5. Next, an *ample* set method is used to generate a sufficient fragment of $L^\kappa = p_1^\kappa \otimes \cdots \otimes p_n^\kappa \otimes \mathcal{DY}$. We prove that the final result of these two steps is *stuttering equivalent* to the full LTS L_f, meaning that it preserves $\mathbf{LTL}_{-\mathbf{X}}$ properties (see, e.g., [CGP00]).

In the following discussions, we write $L^\kappa = (\Sigma^\kappa, s_0^\kappa, A^\kappa, Tr^\kappa)$. Observe that $\Sigma^\kappa = \Sigma_f$, $s_0^\kappa = s_{0\,f}$ and $A^\kappa = A_f \cup \{\kappa_p \mid p \in P\}$ and, given Tr_f, definition 7.5 is used to derive Tr^κ. We can thus partition $\lambda(A^\kappa)$ into three disjoint sets: $\lambda(A^\kappa) = \mathbf{I} \cup \mathbf{S} \cup \mathbf{R}^\kappa$, where $\mathbf{R}^\kappa = \mathbf{R} \cup \{\kappa_p \mid p \in P\}$. All κ actions are added to $\mathbf{R}$, since these actions have to be treated as (artificial) $\mathbf{recv}$ actions. We define the projection function $\aleph : A^\kappa \to \mathcal{P}$ as $\aleph(a) = p$ if $a = \kappa_p$ or $a = a_p(m)$ for some $m \in \Omega(A^\kappa)$. Intuitively, $\aleph(a)$, with a being an action, returns the identity of the process which performs a.

7.6. DEFINITION. *The relation $\sim \subseteq A^\kappa \times A^\kappa$ is defined as $\forall a, a' \in A^\kappa.\ a \sim a' \iff \aleph(a) = \aleph(a')$.*

Clearly $\sim$ is an equivalence relation, and thus partitions A^κ into equivalence classes, such that each class contains only actions performed by one particular process. The set of all equivalence classes in $X \subseteq A^\kappa$ is the *quotient* set of X by $\sim$, and is denoted by $\frac{X}{\sim}$.

For action $a \in A^\kappa$ the equivalence class $[a]$ is defined as $[a] = \{a' \in A^\kappa \mid a \sim a'\}$. At state s, we write $[a]_s = [a] \cap en(s)$. Let $X \subseteq A^\kappa$. We define $\mathbb{V}(X) = \{a \in X \mid \lambda(a) \in \mathbf{V}\}$. Intuitively, $\mathbb{V}(X)$ consists of all the visible internal actions which occur in X. The function $\mathbb{R}$ is similarly defined as $\mathbb{R}(X) = \{a \in X \mid \lambda(a) \in \mathbf{R}^\kappa\}$ (recall that κ_p actions belong to $\mathbf{R}^\kappa$). Functions $\mathbb{I}$ and $\mathbb{S}$ are defined likewise. We let $\partial_\kappa(X) = X \setminus \{\kappa_p \mid p \in \mathcal{P}\}$.

Before defining which requirements the *ample* set in POR for branching security protocols has to satisfy, we note that definition 7.6 can be applied to L_f, simply because $A_f \subseteq A^\kappa$. As mentioned earlier, our POR algorithm receives $p_1^\kappa, \cdots, p_n^\kappa$ as input, so the conditions which are checked for *ample* set refer to this setting (having L^κ in mind). Nevertheless, the final $\mathbf{LTL}_{-\mathbf{X}}$ preserving result is proved with regard to L_f as the full LTS (see theorem 7.2).

7.7. DEFINITION. *At each state s, the set of actions to explore, i.e. $ample(s)$, is constructed in two phases. First, we construct an $ample^0(s)$ set that satisfies the following requirements:*

r0. $ample^0(s) \subseteq en(s)$*; and* $ample^0(s) = \emptyset \implies en(s) = \emptyset$.

r1. For all $a \in en(s)$*, if* $a \in ample^0(s)$*, then* $[a]_s \subseteq ample^0(s)$.

r2. $\mathbb{V}(ample^0(s)) \neq \emptyset \implies en(s) \subseteq ample^0(s)$.

r3. $\mathbb{R}(ample^0(s)) \neq \emptyset \implies en(s) \subseteq ample^0(s)$.

And in the second phase, we define $ample(s) = \partial_\kappa(ample^0(s))$.

Requirement *r0* is a sanity check, cf. condition C0 in theorem 7.1. Requirement *r1* states that if one action of process p is explored, all other enabled actions of p have to be explored as well, since, otherwise, they may disable each other. This requirement was not included in [CJM00a] that only deals with non-branching protocols. Requirements *r2* and *r3*, similar to [CJM00a], prevent prioritising visible internal actions and **recv** actions over any other action, respectively. As κ actions are merely an artificial apparatus to detect the existence of $recv$ actions in case the corresponding **recv** actions are not present in the state space, there is no reason to explore κ actions. Therefore, after constructing $ample^0$, all κ actions are removed from further explorations.

Algorithm 7.2 shows constructing an ample set that meets the requirements of definition 7.7. We emphasise that this algorithm receives $p_1^\kappa, \cdots, p_n^\kappa$ as its input.

```
1:  Open := ∅; Closed := ∅
2:  Open.insert(s_0)
3:  while Open ≠ ∅ do
4:     s := Open.extract
5:     Closed.insert(s)
6:     Construct en(s)/~ (see definition 7.6) and name its elements as c_1, ..., c_ℓ.
7:     T := {c_i, i ∈ {1, ..., ℓ} | V(c_i) ∪ R(c_i) = ∅}
8:     if T ≠ ∅ then
9:        Pick a c ∈ T
10:       ample(s) := c
11:    else if T = ∅ then
12:       ample(s) := ∂_κ(en(s))
13:    for all a ∈ ample(s) do
14:       if a(s) ∉ Closed ∧ a(s) ∉ Open then
15:          Open.insert(a(s))
```

Algorithm 7.2: *POR for branching security protocols*

Preserving $\mathbf{LTL_{-X}}$ properties

Below, we show that for branching security protocols any POR algorithm that meets the requirements of definition 7.7 preserves $\mathbf{LTL_{-X}}$ properties. We start with a few auxiliary lemmas.

7.1. LEMMA. *In L_f, and similarly in L^κ, For any two actions a and a', we have $[a] \neq [a'] \implies (a, a') \in IND$.*

Proof: Since $\aleph(a) \neq \aleph(a')$, clearly a and a' are performed by different processes. Let us assume that $a, a' \in en(s)$, for some $s \in \Sigma_f$. We distinguish the following cases:

- $\lambda(a), \lambda(a') \in \mathbf{I} \cup \mathbf{S}$: In this case, if $s \xrightarrow{a'} s_1$, then $a \in en(s_1)$. Similarly, if $s \xrightarrow{a} s_2$, then $a' \in en(s_2)$. These clearly follow from definition 7.3.
- $\lambda(a), \lambda(a') \in \mathbf{I} \cup \mathbf{R}$: Similar to the previous case.
- $\lambda(a) \in \mathbf{S}$ and $\lambda(a') \in \mathbf{R}$: Note that $K(s) \vdash \Omega(a')$. Therefore, the intruder does not require the information contained in $\Omega(a)$ to construct $\Omega(a')$. Otherwise, these would not be enabled in the same state (recall that $a, a' \in en(s)$). If $s \xrightarrow{a'} s_1$, then definition 7.3 implies that $a \in en(s_1)$. Now, we show that if $s \xrightarrow{a} s_2$, then $a' \in en(s_2)$. Observe that $K(s) \subseteq K(s_2)$, therefore, according to the monotonicity of $\vdash$, we have $K(s_2) \vdash \Omega(a')$, hence $a' \in en(s_2)$.

Moreover, if $s \xrightarrow{a.a'} s'$ and $s \xrightarrow{a'.a} s''$, then $s' = s''$. This is because performing a has no effect on the state of the process that performs a' and vice versa, as processes do not directly communicate with each other, but only via the intruder. This completes the proof. ■

7.2. LEMMA. *Assume $\mathbb{R}([a]_s) = \emptyset$ in L^κ. Then, in L_f, $s \xrightarrow{a'} s'$ with $[a] \neq [a']$, implies $[a]_s = [a]_{s'}$.*

Proof: Consider L_f. Since $\aleph(a) \neq \aleph(a')$, according to lemma 7.1, $\forall b \in [a].\ (b, a') \in IND$, and consequently, for any $b \in [a]$, $s \xrightarrow{b} s_1$ and $s \xrightarrow{a'} s'$ imply that $\exists s_2.\ s' \xrightarrow{b} s_2$. Therefore $[a]_s \subseteq [a]_{s'}$. Now, we need to show that $[a]_{s'} \subseteq [a]_s$. Note that the enabledness of actions of $\mathbf{I}$ and $\mathbf{S}$ only depend on the state of the process performing them (see definition 7.3). As the state of $\aleph(a)$ is the same in s and s' (because $s \xrightarrow{a'} s'$ and $\aleph(a) \neq \aleph(a')$), $\mathbb{I}([a]_s) = \mathbb{I}([a]_{s'})$ and similarly for $\mathbb{S}$. However, actions of $\mathbf{R}$ also depend on the state of the intruder, i.e. when a process is waiting to receive a message, the intruder's knowledge determines what messages, if any, can be sent to that process. Here, $\mathbb{R}([a]_s) = \emptyset$ in L^κ implies that in particular $\kappa_p \notin en(s)$, with $p = \aleph(a)$, in L^κ (recall that $\forall p \in \mathcal{P}.\ \kappa_p \in \mathbf{R}^\kappa$ in L^κ). Therefore, κ_p is not enabled at s' in L^κ, as well. As a result, $\mathbb{R}([a]_{s'}) = \emptyset$ in L^κ. This implies that also in L_f, $\mathbb{R}([a]_{s'}) = \emptyset$, completing the proof. ■

7.1. PROPOSITION. *Let $a \in en(s)$ (in L_f and L^κ) and assume that $\mathbb{R}([a]_s) = \emptyset$ in L^κ. For all traces $\alpha = \alpha_1 \cdots \alpha_i \cdots$, such that $\alpha \in \pi(s)$ in L_f, we have*

$$\forall i.\ \Big((a, \alpha_i) \in \mathit{DEP} \implies \exists j \leq i.\ \alpha_j \in [a]_s\Big)$$

Proof: Let $(\alpha_i, a) \in \mathit{DEP}$, for some i. According to lemma 7.1, $(a, \alpha_i) \in \mathit{DEP} \implies \alpha_i \in [a]$. Two cases are possible here:

- If $i = 1$, then obviously letting $j = i$ completes the proof.
- If $i > 1$, assume $\forall j < i.\ \alpha_j \notin [a]_s$. We prove that then $\alpha_i \in [a]_s$. The assumption in particular implies that $\alpha_1 \notin [a]_s$. Since $\alpha_1 \in en(s)$, we deduce that $\alpha_1 \notin [a]$. Therefore, according to lemma 7.1, $(a, \alpha_1) \in \mathit{IND}$. As $\mathbb{R}([a]_s) = \emptyset$ in L^κ, lemma 7.2 implies that $[a]_s = [a]_{s_1^\alpha}$ in L_f (recall that the sequence of states associated to α is denoted s^α, see § 4.1.1). Repeating this argument, we can show that $[a]_{s_1^\alpha} = [a]_{s_2^\alpha}, \ldots$, and finally $[a]_{s_{i-1}^\alpha} = [a]_{s_i^\alpha}$ in L_f. Hence $[a]_s = [a]_{s_i^\alpha}$. Since $\alpha_i \in [a]$, and clearly $\alpha_i \in [a]_{s_i^\alpha}$, it follows that $\alpha_i \in [a]_s$.

This completes our proof. ■

Now we are ready to prove the main theorem about our POR algorithm. Given is specification of a branching security protocol $p_1, \cdots, p_n$, with $L_f = p_1 \otimes \cdots \otimes p_n \otimes \mathcal{DY}$. Let por be a POR algorithm in which *ample* sets satisfy the conditions of definition 7.7. We write L_{por} for the LTS generated by applying por on $p_1^\kappa, \cdots, p_n^\kappa$.

7.2. THEOREM. $\forall \phi \in \mathbf{LTL}_{-\mathbf{X}}.\ L_f \models \phi \iff L_{\mathsf{por}} \models \phi$.

Proof: We show that the conditions C_0, C_1, C_2 and C_3 of theorem 7.1 hold for our proposed *ample* set. Condition C_0 holds because of r_0. Condition C_1 holds because of proposition 7.1. Condition C_2 holds because of r_2 and r_3. Recall that we assume that only members of $\mathbf{V} \cup \mathbf{R}$, which is equivalent to $\mathbf{V} \cup \partial_\kappa(\mathbf{R}^\kappa)$, appear in the properties being verified. Condition C_3 holds simply because we consider acyclic security protocols. ■

7.4 POR in constraint solving

Constraint solving for analysing security protocols has mostly been used in verifying non-branching protocols. However, recently the constraint solving approach of [MS01] has been extended to a large class of branching security protocols, namely contract signing protocols [KK05]. In the previous sections we focused on POR algorithms for explicit state model checking settings. Below we sketch how our POR algorithm can be used in the constraint solving setting.

We first cast the constraint solving approaches of [MS01] and [KK05] to our formalism. Given a specification $p_1, \cdots, p_n$, we construct the product of the LTSs $p_1 \times \cdots \times p_n$, where $p \times q$ is defined in the following.

7.8. DEFINITION. *Given $p = (\Sigma_p, s_{0_p}, A_p, Tr_p)$ and $q = (\Sigma_q, s_{0_q}, A_q, Tr_q)$, $p \times q$ is the LTS $(\Sigma_p \times \Sigma_q, (s_{0_p}, s_{0_q}), A_p \cup A_q, Tr_{p\times q})$, where $Tr_{p\times q}$ is*

$$\begin{aligned} Tr_{p\times q} = \{(s_p, s_q) \xrightarrow{a} (s'_p, s_q) \mid (s_p, a, s'_p) \in Tr_p\} \cup \\ \{(s_p, s_q) \xrightarrow{a} (s_p, s'_q) \mid (s_q, a, s'_q) \in Tr_q\} \end{aligned}$$

Note that p and q do not communicate directly (only indirectly via the intruder). The resulting $p \times q$ can be seen as an uninstantiated state space (i.e. the actions of $A_{p\times q}$ contain messages with uninstantiated variables). In constraint solving approaches, every maximal trace of $p_1 \times \cdots \times p_n$ is examined by a constraint solving algorithm CS to decide whether $\mathcal{DY}$ can trick the processes of $\mathcal{P}$ to execute this path and, if so, whether it constitutes an attack or not. We note that algorithm 7.2 can readily be used in generating a sufficient fragment of the uninstantiated state space $p_1 \times \cdots \times p_n$, when the depth-first exploration strategy is adopted.

We remark that in this scenario the infinitely branching behaviour of the $\mathcal{DY}$ intruder is captured in the CS phase, cf. [MS01, KK05]. Moreover, calculating κ-translations can altogether be omitted from the POR algorithm in this setting. This is because in the generation phase, no $recv_p$ action will be hidden as a result of the intruder's lack of knowledge: The intruder's abilities are modelled in the later phase of CS.

For other optimisation techniques regarding constraint solving for security protocols we refer to [MS01, CE02, BMV03].

7.5 Experimental results

The proposed POR algorithm (algorithm 7.2) only relies on local tests, namely $ample(s)$ can be constructed with inspecting $en(s)$ at each state, and no cycle detection mechanism is required in our setting. This implies that a distributed implementation of the algorithm can be very efficient, as no POR-specific inter-workstation communications would be required. Below, we report some experimental results based on a distributed breadth-first implementation of algorithm 7.2, hereafter call DPOR, in the distributed μCRL tool-set [BFG$^+$01, BCL$^+$07].

In the distributed setting, usually, a number of *client* machines work together to generate the state space, while a *manager* process keeps track of their progress. These machines are connected with an asynchronous communication network. Algorithm 7.4 shows the part of our DPOR algorithm which is ran by each client machine. The manager decides to terminate the generation, when no new states are found. The code for the manager process is presented in algorithm 7.3. We refer to [Wij07] for details of distributed state space generation.

In algorithm 7.4, we have included standard constructs for distributed state space generation: Each client process has a unique identity ID and maintains its own local vari-

```
Require: set of IDs
 1: repeat
 2:   NextLevel := F
 3:   for all ID ∈ set of IDs do
 4:     RecvFromClientNewStatesFound(ID, v)
 5:     NextLevel := NextLevel ∨ v
 6:   if ¬NextLevel then
 7:     SendToClients(finish)
 8:   else
 9:     SendToClients(¬finish)
10: until ¬NextLevel
```

Algorithm 7.3: *Manager process in DPOR*

ables $Open$, $Closed$, etc. Client processes are also provided with a hash function $\#$ that assigns to each state a unique *owner*, which is the process responsible for its expansion. The procedure SendToClientsNextLevel(S), with S being a set of states, sends the states of S to their corresponding owners (determined by $\#$). Conversely, the procedure $S :=$ ReceiveFromClientsNextLevel() receives from all clients the states that are to be processed by the current client and returns them in the set S. The procedures RecvFromMgr() and SendToMgrNewStatesFound(ID, $|S| > 0$) are used to communicate with the manager. The procedure RecvFromMgr() asks the manager if the client should continue the generation and SendToMgrNewStatesFound sends a Boolean value (i.e. if $|S| > 0$) to the manager. This informs the manager whether the client has found any new state in the current round or not. The counterparts of these commands appear in the manager process, algorithm 7.3. Note that, via communicating with the manager, the clients synchronise their progress.

We remark that the purpose of using the $Closed$ set in algorithm 7.4 is to avoid state revisits, and it is not needed to guarantee the termination of the algorithm, if the LTSs are acyclic. Therefore, this set can gradually be removed from memory and be stored on high latency media (e.g. disks), in case memory limits are reached, without endangering the termination of the algorithm. For similar approaches to memory management see [BLP03, HW06].

7.5.1 A case study

To demonstrate the effectiveness of the proposed POR algorithm we have modelled a Digital Rights Management (DRM) protocol, described in detail in [TKJ07], see also § 6.3. Below, we briefly describe this protocol and our experimental results using an implementation of DPOR algorithm 7.4 in the distributed μCRL tool-set.

The protocol of [TKJ07] comprises a finite set of trusted content rendering devices C and a finite set of trusted entities T. The goal of the protocol is to provide a secure environment for fair exchange of digital items among the members of C in peer-to-peer exchanges, and

Require: ID , $\# : \Sigma \rightarrow$ set of IDs
1: $Open := \emptyset$; $Closed := \emptyset$
2: **if** $\#(s_0) =$ ID **then**
3: $Open := \{s_0\}$
4: **repeat**
5: $Next := \emptyset$
6: **for all** $s \in Open \setminus Closed$ **do**
7: Construct $\frac{en(s)}{\sim}$ (see definition 7.6) and name its elements as $c_1, \ldots, c_\ell$.
8: $T := \{c_i, i \in \{1, \ldots, \ell\} \mid \mathbb{V}(c_i) \cup \mathbb{R}(c_i) = \emptyset\}$
9: **if** $T \neq \emptyset$ **then**
10: Pick a $c \in T$
11: $ample(s) := c$
12: **else if** $T = \emptyset$ **then**
13: $ample(s) := \partial_\kappa(en(s))$
14: **for all** $a \in ample(s)$ **do**
15: $Next := Next \cup\ a(s)$
16: SendToMgrNewStatesFound(ID, $|Next| > 0$)
17: $command :=$ RecvFromMgr()
18: **if** $command \neq$ finish **then**
19: SendToClientsNextLevel($Next$)
20: $Closed := Closed \cup Open$
21: $Open :=$ ReceiveFromClientsNextLevel()
22: **until** $command =$ finish

Algorithm 7.4: *Breadth-first DPOR for branching security protocols*

between a member of C and a member of T in the so called *direct purchases*. The attacker is external to $C \cup T$, as these are all trusted. It may however *own* some of the content rendering devices, and thus deliberately operate them. In case of malicious acts or excessive delays, the wronged (trusted) device can resort to one of the trusted entities (hence having choice points). The finite set D contains the items available to the trusted devices in the protocol. Each item is bundled with a right declaring the terms of use of that particular item. The set of rights is denoted R. To keep the state space finite, each $c \in C$ has access to a finite set N_c of nonces to start fresh exchanges.

Design goals

The design goals of the protocol include secrecy of the exchanged contents, resisting *content masquerading* and providing fair exchange [TKJ07]. Below, we briefly explain and, then,

express these properties in $\mathbf{LTL}_{-\mathbf{X}}$. [7]

- Secrecy requires that no element of D is ever revealed to the intruder. We thus allow the intruder to perform an internal action $\mathrm{reveal}_{\mathcal{DY}}(d)$, [8] once $d \in D$ can be derived from its knowledge, i.e. when $\Gamma(s) \vdash d$, this action becomes available to the intruder, see § 7.2.1. The following property formalises secrecy.

  $$\forall d \in D.\ \neg\Diamond(\mathrm{reveal}_{\mathcal{DY}}(d))$$

  Note that since D is finite, $\forall d \in D$ can be rewritten into a finite number of conjunctions. Therefore, $\forall$ is merely a syntactic shorthand and does not go beyond $\mathbf{LTL}_{-\mathbf{X}}$.

- Content masquerading happens when content d_1 is passed off to a trusted device as content d_2, with $d_1 \neq d_2$. A way to resist content masquerading is to ensure that if a trusted device does not request d, then the intruder can never feed it with d.

  $$\forall d \in D, p \in \mathcal{P}.\ \neg\ ((\neg\mathrm{request}_p(d))\mathcal{U}\mathrm{update}_p(d))$$

  Actions $\mathrm{request}_p$ and update_p represent the points where p requests a content and updates its local set of right-content bundles with a received content, respectively.

- Fairness in exchange requires that if p, a trusted device, pays q for a content, it will eventually receive it (see also § 2.2). Below, $price(d)$, with $d \in D$, is the price tag associated with d. The term $p.q.price(d).d$ denotes a payment order from p to q, for the amount $price(d)$, related to the exchange of item d. The actions $make$ and $cash$ have their intuitive meaning.

  $$\begin{aligned}\forall d \in D, p, q \in \mathcal{P}.\ \Box(make_p(p.q.price(d).d)) \implies\ & \\ \neg\Diamond cash_q(p.q.price(d).d)& \\ \vee\ \Diamond\mathrm{update}_p(d))&\end{aligned}$$

The setting of the experiment

In our experiments, we have used machines with a single 64 bit Athlon 2.2 GHz CPU and 1 GB RAM, running Linux FEDORA CORE 6, connected with Gigabit Ethernet (1Gbps). In the following, measured time refers to elapsed time (wall clock time), that is the time taken from the beginning till the end of an experiment. Therefore, this is not only computation time, but also reflects periods of waiting, etc. The experiments were performed using the μCRL tool-set version 2.17.13, which implements the DPOR algorithm. [9]

[7] To avoid cluttering the properties, various cases where a content is traded with a peer device or a trusted provider, or when an intruder-fabricated right is attached to a content, are ignored here. A detailed analysis would also contain resisting right masquerading, etc., cf. [TKJ07].

[8] Compare with lemma 6.1.

[9] The μCRL tool-set is available at `http://www.cwi.nl/~mcrl/`, under the GNU GPL conditions.

Table 7.1: *Effectiveness of DPOR. (Time is in min:sec format.)*

Instance		Exhaustive		DPOR		Reduction	
$\|N_c\|$	$\|T\|$	# States	Time	# States	Time	in # States	in Time
1	2	89,155	02:27.74	45,871	01:56.49	48.5%	21.2%
2	2	277,459	06:23.82	145,559	05:21.97	47.5%	16.1%
1	3	2,674,940	52:47.48	1,082,122	34:09.28	59.5%	35.3%
2	3	11,896,384	269:08.64	4,794,745	169:40.64	59.7%	36.9%

Effectiveness of DPOR

In this section, we discuss the effectiveness of our DPOR algorithm by comparing the number of states and generation time when using the DPOR algorithm versus exhaustive distributed breadth-first state space generation (both implemented in the μCRL tool-set). In table 7.1, we consider four instances of the DRM protocol described above, with$|C| = 2$ and $|D| = |R| = 1$. The number of fresh nonces available to each rendering device is denoted by $|N_c|$, and $|T|$ denotes the number of concurrent trusted party processes. The time column shows the amount of time required by 16 machines (15 clients, and one manager) to complete the generation task. It is worth mentioning that state expansion is in general time consuming in security protocols, as it involves matching the messages that protocol participants can receive to the messages that the attacker can construct.

In table 7.1, we observe that for large state spaces approximately a 60% reduction in the number of states is achieved. Although DPOR loads the generation algorithm with extra computation, the gained reduction definitely compensates for it, as is evident from the time columns.

Some common characteristics of these state spaces are reported in table 7.2. Given a finite LTS $L = (\Sigma, s_0, A, Tr)$, for $s \in \Sigma$, we define $d(s) = \min_i\{(s_0, s) \in \rightarrow^i\}$, where $\rightarrow^i = \rightarrow \times \rightarrow^{i-1}$ and $\rightarrow^1 = \cup_{a\in A} \stackrel{a}{\rightarrow}$ (see the notations of § 4.1.1). The *depth* of L is $\max\{d(s) \mid s \in \Sigma\}$. The *average fan-out* of L is computed as $\frac{|Tr|}{|\Sigma|}$. Note that the depth column appears once in table 7.2 because our POR algorithm does not change the depth. We observe that in all the experiments, the average fan-out of the reduced state space is less than the full one. See, e.g., [Pel04] on how these parameters correlate to the time and memory typically used in state space generation.

The results of table 7.1 unfortunately cannot readily be compared with existing tools implementing POR for security protocols, e.g. [CJM00a, CM05]. This is because these tools do not deal with branching security protocols. Going back to non-branching protocols, our algorithm coincides with the algorithm of [CJM00a]. We expect the algorithm of [CM05] to yield more reductions, compared to our algorithm, when analysing non-branching protocols. This is because the algorithm in [CM05] has been optimised for a rather narrow subset of $\mathbf{LTL}_{-\mathbf{X}}$, which is the class of properties preserved by our POR algorithm. See also § 7.6.

Table 7.2: *State space characteristics.*

Instance		Depth	Average Fan-out	
$\|N_c\|$	$\|T\|$		Full state space	Reduced state space
1	2	24	2.85	1.40
2	2	36	2.74	1.38
1	3	27	3.44	1.48
2	3	47	3.36	1.46

Table 7.3: *Scalability of DPOR. (Time is in min:sec format.)*

Instance \ # Machines		1	4	8	16
$\|N_c\|$	$\|T\|$	Time	Time	Time	Time
1	2	11:56.44	04:45.27	03:16.72	01:56.49
2	2	43:14.15	15:54.48	10:15.75	05:21.97
1	3	374:33.38	123:44.43	71:45.94	34:09.28
2	3	2676:34.70	1286:15.36	356:18.81	168:05.45

Scalability of DPOR

In table 7.3, we compare the generation time required by DPOR using different numbers of machines. As was explained earlier, since the DPOR algorithm does not require extra communications to synchronise on the POR pruning part, we expect it to scale up well. Figure 7.5 shows the results of this table on log-scale graphs. These measurements indeed confirm that DPOR exhibits reasonable scalability.

A phenomenon that can be observed in the case of $(|N_c|, |T|) = (2, 3)$, is that the speed-up factor [10] for 8 and 16 machines experiments are absurdly high. Although reasoning based on the speed-up factor has some defects (e.g. see [Cro94]), nonetheless, we find it illuminating to discuss why this irregularity appears.

Through the experiments on the (2,3) case, we witnessed that when using 1 and 4 machines, the available RAM of the client machines is not enough (distributed implementation is often motivated because of memory limitations of single machines). Therefore, the operating system starts swapping, i.e. using high-latency disk memory besides RAM. Therefore, these experiments take much more time than expected. For instance, the reduced state space of the (2,3) case is roughly 5 times larger than the reduced state space of the (1,3) case (see table 7.1), whereas table 7.3 shows that when using 1 and 4 machines, the (2,3) case is about nine times slower than the (1,3) case. This is due to the time penalty imposed by swapping.

[10] Speed-up factor here is defined as $\frac{t_1}{t_n}$, where t_1 is the time required by one machine to perform the generation using the parallel algorithm, and t_n is the time required by n parallel machines to perform the same job.

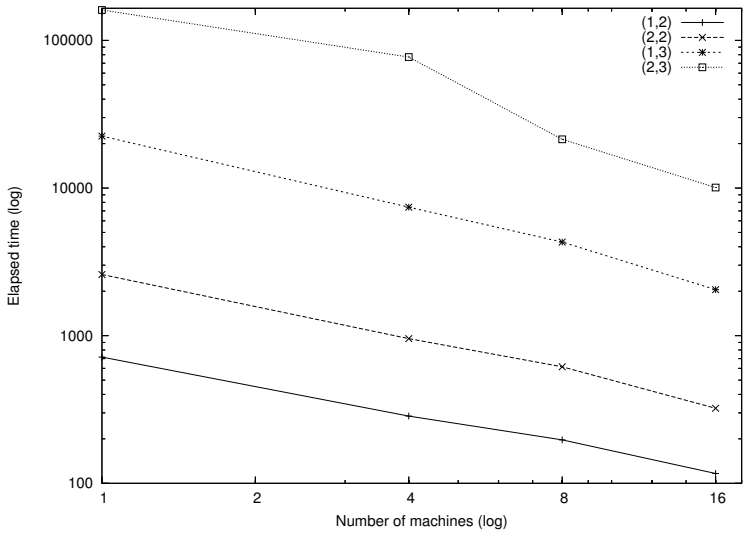


Figure 7.5: *Elapsed time in DPOR*

7.6 Conclusions, related and future work

In this chapter we have extended the POR algorithm of [CJM00a] to branching security protocols. The proposed algorithm has been implemented in the μCRL general-purpose state space generation tool-set. Our POR algorithm is in fact built upon a *priority beam search* mechanism which has recently been added to the μCRL framework [TW07b, Wij07].

Related work

Algorithms for POR are an established branch of model checking and state space generation fields, see, e.g., [PPH97, Pel98, CGP00]. As finding an optimal $ample$ set is NP-hard [Pel93], many POR algorithms in the literature focus on a particular setting and propose heuristics for computing a (near-)optimal $ample$ set in that setting, e.g. see [Pel97]. Our work can be seen as one of these heuristics, which preserves $\mathbf{LTL}_{-\mathbf{X}}$ for a large class of security protocols.

When it comes to distributed POR, a major issue is to find an efficient way to satisfy the *cycle condition*, see C3 in theorem 7.1. Among notable work which tackle this problem in the distributed setting are [PG02, BČMŠ05]. Our work is in a sense orthogonal to this work, as the cycle condition is irrelevant in cycle-free state spaces, which is our framework. In contrast, we propose an algorithm to efficiently find a suitable subset of $en(s)$ to be explored at each state s, with solely local inspections. See also our future work.

POR techniques for analysing security protocols can perhaps be traced back to [SS98],

where some methods to reduce the number of states when verifying security protocols are proposed, but no formalisation of the techniques is provided. Similarly, [Bas99] applies heuristics to prune the search space of security protocols, but only intuitively justifies them.

Clarke et al. [CJM00a] were the first to present a POR algorithm for security protocols and formally prove its properties. Their algorithm is tailored for non-branching protocols. The focus of this chapter has been on extending this algorithm to branching protocols. The logic that is used in [CJM00a] is in general incomparable to $\mathbf{LTL}_{-\mathbf{X}}$. It is a subset of $\mathbf{LTL}_{-\mathbf{X}}$ in the sense that it only allows past time modal operator $\Diamond^{-1}$, which can be simulated in $\mathbf{LTL}_{-\mathbf{X}}$ (see, e.g., [LMS02]). [11] We however note that since models of their logic are acyclic finite Kripke structures, any formulae of $\mathbf{LTL}_{-\mathbf{X}}$ can, with some overhead, be expressed in terms of the $\Diamond^{-1}$ operator. The logic of [CJM00a] is nonetheless richer than $\mathbf{LTL}_{-\mathbf{X}}$ in that it explicitly allows using epistemic operators. For instance, they can specify properties such as $\mathcal{E}_{\mathcal{DY}}(m) \implies \neg\exists p \in \mathcal{P}.\ \Diamond^{-1}\mathcal{E}_p(m)$, where $m \in Msg^c$ and $\mathcal{E}_x(m)$ states that process x *knows* m, that is $K_x \vdash m$ with K_x being the knowledge set of process x (cf. § 7.2.1). The aforementioned property thus intuitively stipulates that the knowledge of honest processes is never shared with the intruder. In order for their POR algorithm to preserve this logic, it is required that terms which refer to the intruder's knowledge appear only negatively in the properties. To see why, consider the property $\phi = (\mathcal{E}_p(t_0) \implies \Diamond^{-1}\mathcal{E}_{\mathcal{DY}}(t_1))$, stating that if the honest process p knows t_0, then in some past time the intruder knew t_1. Now, consider processes $p = recv_p(v).\delta$, with $v \in MV$, and $q = send_q(t_1).\delta$, and consider the system $L_f = p \otimes q \otimes \mathcal{DY}$, such that $\Gamma_0 \vdash t_0$, but $\Gamma_0 \nvdash t_1$ (recall the definitions of § 4.1). If $\mathbf{send}_q$ is prioritised over $\mathbf{recv}_p$, the property ϕ would hold in the reduced LTS, while this is not the case in L_f, thus ϕ is not preserved. We do not need to put such a constraint on our POR algorithm, since epistemic operators are not allowed to appear explicitly in $\mathbf{LTL}_{-\mathbf{X}}$.

In [CM05], a POR algorithm for non-branching security protocols has been proposed that gains more reductions compared to [CJM00a], but preserves a narrow subset of $\mathbf{LTL}_{-\mathbf{X}}$ properties: Only properties of the form $\Diamond\mathcal{E}_{\mathcal{DY}}(t)$, where $t \in Msg^c$, are considered in [CM05].

Future work

Extending our POR algorithm to cover cyclic processes seems to be possible using standard techniques to handle cycles in POR algorithms, as presented in, e.g., [CGP00]. This however needs to be fully investigated.

It must be interesting to experiment with various classes of security protocols to assess the effectiveness of DPOR in different settings. Our current observations on optimistic FE protocols suggest that in many cases the *ample* sets computed using our POR algorithm are singletons. This can potentially be exploited to devise a nearly as effective POR algorithm, which preserves the branching logic $\mathbf{CTL}^*_{-\mathbf{X}}$. This can potentially be used to check branching security properties, such as anonymity.

[11] Given a trace $\alpha = \alpha_1 \cdots \alpha_i \cdots$, let $\alpha^{i\infty} \models \Diamond^{-1}\phi$ iff $\exists j.\ 1 \leq j \leq i \wedge \alpha^{j\infty} \models \phi$.

A third direction for future investigations is related to POR with *deferral*. Deferring certain actions can apparently be of use in generating state spaces for checking security-related properties. For instance, consider the authentication property $\phi = \Box(auth_p(q) \implies \Diamond^{-1} auth_q(p))$, interpreted as: If p authenticates q, then in some past time q should have authenticated p as well. We contend that in generating acyclic state spaces of authentication protocols (when checking ϕ), action $auth_q(p)$ can be postponed, i.e. not explored as long as possible. Let α be a trace in the LTS that is to be checked against ϕ. We consider two cases:

- If action $auth_q(p)$ appears before $auth_p(q)$ in α, then clearly $\alpha \in \phi$. Therefore, exploring $auth_q(p)$ actions early is useless to our purpose, which is looking for counterexamples for ϕ.
- Else, either $auth_q(p)$ does not appear in α at all, or $auth_q(p)$ appears after $auth_p(q)$ in α. These two situations are not distinguished by ϕ, namely in both cases $\alpha \notin \phi$.

Therefore, we can always postpone taking $auth_q(p)$ while generating state spaces for checking ϕ. As mentioned above, our POR tool-set is built upon a priority beam search mechanism. Therefore, negative priorities, which can in principle implement deferrals, are available to our tool-set. The theoretical basis of such reductions is however yet to be investigated.

Chapter 8

Concluding remarks

We conclude the thesis with recapitulating our main contributions and, then, pointing out some future research directions.

This thesis concerns *design* and *formal analysis* of optimistic fair exchange protocols. In § 3, we design a novel fair certified email protocol. A certified email protocol enables Alice to send an email to Bob in exchange for a receipt. The receipt is a proof that shows Bob has received the email. A fair certified email protocol guarantees fairness in this exchange: Bob receives the email if and only if Alice receives the receipt. Optimistic fair exchange protocols (including our certified email protocol), if deployed in asynchronous settings such as the Internet, require stateful trusted judicators. Intuitively, the exchange partners are provided with fallback scenarios, in which they can resolve a dispute at the judicator. To maintain fairness, the judicator needs to keep certain information about each resolved dispute, virtually for an indefinite amount of time. The novelty in our proposed protocol is to reduce the amount of the storage that the judicator requires, using forward hash chains. The idea is to establish a forward (double) chain of keys to secure the exchanges between Alice and Bob, and store only the seed of the chain at the judicator. The judicator can derive the required keys at will, only using the seed.

Design and formal analysis of protocols are in fact tightly related. Designing a protocol requires a clear understanding of the goals that are to be achieved. In this early phase, formal methods can help us with clarifying the goals, determining the class of achievable goals, etc. After a protocol has been designed, it is desirable to validate the design against its goals. Formal methods provide us with the required tools and techniques to tackle this problem as well. In short, before starting to design a protocol and after completing the design, formal methods can be of great use.

In our experience, formal methods have also been helpful while designing protocols. Precisely specifying the behaviour of the protocol participants in formal languages brings hidden assumptions and difficulties to the front. This helps developers to more aptly gear the design towards its goals.

The above discussions mainly concern qualitative properties of security protocols, such as preserving secrecy. In verifying quantitative aspects of protocols (for example, various efficiency measures), formal methods are still in their primary development stages. Nevertheless, our experience shows that model checking can be used to minimise a security protocol in a *safe* manner, i.e. the minimised protocol satisfies the desired (qualitative) properties and, be-

sides, it contains only message terms which, if removed, endanger either the functionality or the security of the protocol.

Using formal methods has its own downside. A first issue is the risk of oversimplifying protocols. Formal techniques provide insightful information about a protocol's *model*, not the protocol's implementation. Therefore, wrong modelling decisions can lead to unsound analyses. This is in fact a major problem, since modelling a protocol in its full detail is often too convoluted and expensive to be practical. In this thesis, we have carefully studied how formal models may soundly reflect resilient channels. Resilient channels are abundantly used in optimistic fair exchange protocols and, thus, appear in their models as well.

According to Asokan, "a message inserted into a resilient channel will eventually be delivered" [Aso98]. A natural way to encode a resilient channel into a protocol model is to devise a fairness constraint which excludes all the executions that violate this condition. In § 5, we show that such a fairness constraint turns out to be complicated, in the following sense: It is not locally testable, i.e. to tell whether an execution is fair or not, it is required to look arbitrarily deep into the history of the execution, and, moreover, the fairness constraint depends on the contents of the messages that are transmitted in the protocol. As complex fairness constraints are not suitable for automatic verification techniques (such as model checking), we propose a modified intruder model and a much simpler fairness constraint, which implement the desired resilience condition. The proposed simple fairness constraint is indeed locally testable, and does not depend on the message contents.

Our study shows that most existing formal analyses which incorporate resilient channels represent them in an *unsound* manner. Namely, there are protocols which do not achieve their goals in any environment that provides resilient channels, while, in contrast, the formal models of these protocols satisfy the goals. [1] This of course does not immediately refute the results already established by these analyses. This however shows that such results cannot be readily related to the real world implementations of protocols in the presence of resilient channels. This underlines the difficulties in representing practical security concepts soundly in their models.

Another major problem in using formal techniques is the expertise that is required to use them. Model checking and constraint solving have provided partial solutions to this problem in analysing security protocols, by automating a large part of the analysis. Once the model and the desired properties of a protocol are described in proper formal languages, verifying whether the model satisfies the properties or not, is done with negligible human intervention. These techniques, however, typically suffer from the *state space explosion* problem. Roughly speaking, to verify the properties, the models have to be unfolded. The resulting unfolded models in most practical cases tend to become prohibitively large, thus making the analysis costly in terms of memory and time. To tackle this problem, we have extended an existing partial order reduction technique to optimistic fair exchange protocols. This essentially makes larger protocol models with more details amenable to automatic formal verification.

[1] We remark that these "counterexample" protocols are not necessarily abundant in practice.

Some future research directions

Regarding our certified email protocol (proposed in § 3), we note that since forward key chains are by definition infinite, the state space of (even the smallest instantiation of) the protocol becomes infinite. Devising sound and complete abstraction techniques (such as those presented in [PV04]) to make automated verification of the protocol possible is our next step.

On the formalisation side, there are plenty of unknowns to us. Our formalisation of resilient channels aims at verifying liveness security properties. Fairness in exchange is a liveness security requirements, and so is abuse-freeness (see, e.g., [GJM99]). It would be interesting to see how our results can be applied to the analysis of abuse-free protocols. A more fundamental question concerns extending our formal model to infinitely branching LTSs, which result from security protocols without assuming strongly typed messages. We believe it is possible to lift our results to such LTSs. Another relevant research question is characterising the weakest form of resilience condition under which a certain class of liveness properties hold for security protocols.

An alternative approach to formalise resilient channels, and analyse liveness, is to use partial order semantics (such as strand spaces [JHG99]), instead of the total order semantics that we have adopted in the thesis. How this would affect our formalisation and optimisation techniques is an intriguing issue, yet to be studied.

Related to the partial order reduction section, it would be interesting to extend the proposed algorithm to facilitate checking properties with the fairness constraints devised in § 5.

A question which is of, perhaps only, theoretical interest is precisely determining the relations between the fair exchange problem and its cousins in the distributed computing world, namely distributed consensus and atomic commit problems. Some first results have already been established in this direction, e.g. see [PG99, AFG$^+$04].

Bibliography

[AB03] M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email. In *SAS '03*, volume 2694 of *LNCS*, pages 316–335. Springer, 2003.

[ABC+98] R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. *SIGOPS Oper. Syst. Rev.*, 32(4):9–20, 1998.

[ACC07] A. Armando, R. Carbone, and L. Compagna. LTL model checking for security protocols. In *CSF '07*, pages 385–396. IEEE CS, 2007.

[AdG01] G. Ateniese, B. de Medeiros, and M. Goodrich. TRICERT: A distributed certified email scheme. In *NDSS '01*. Internet Society, 2001.

[ADGH06] I. Abraham, D. Dolev, R. Gonen, and J. Halpern. Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In *PODC '06*, pages 53–62. ACM Press, 2006.

[AFG+04] G. Avoine, F. Freiling, R. Guerraoui, K. Kursawe, S. Vaudenay, and M. Vukolic. Reducing fair exchange to atomic commit. Technical Report 200411, EPFL, Lausanne, Switzerland, 2004.

[AG97] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *CCS '97*, pages 36–47. ACM Press, 1997.

[AG02] M. Abadi and N. Glew. Certified email with a light on-line trusted third party: Design and implementation. In *WWW '02*, pages 387–395. ACM Press, 2002.

[AGGV05] G. Avoine, F. Gärtner, R. Guerraoui, and M. Vukolic. Gracefully degrading fair exchange with security modules. In *EDCC '05*, volume 3463 of *LNCS*, pages 55–71. Springer, 2005.

[AH03] R. Alur and T. Henzinger. Computer-aided verification, 2003. Book chapter available at `http://www.eecs.berkeley.edu/~tah/294-1/9.ps`.

[ALV01] R. Amadio, D. Lugiez, and V. Vanackére. On the symbolic reduction of processes with cryptographic functions. Technical Report 4147, INRIA, France, Mar. 2001.

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Softw. Eng.*, 22(1):6–15, 1996.

[And92] R. Anderson. UEPS - a second generation electronic wallet. In *ESORICS '92*, volume 648 of *LNCS*, pages 411–418. Springer, 1992.

[AR00] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *TCS '00*, volume 1872 of *LNCS*, pages 3–22. Springer, 2000.

[AS84] B. Alpern and F. Schneider. Defining liveness. Technical Report TR 85-650, Dept. of Computer Science, Cornell University, Oct. 1984.

[Aso98] N. Asokan. *Fairness in electronic commerce*. PhD thesis, University of Waterloo, 1998.

[ASW96] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for multi-party fair exchange. Research Report RZ 2892 (# 90840), IBM Zürich Research Lab, Dec. 1996.

[ASW97] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *CCS '97*, pages 8–17. ACM Press, 1997.

[ASW98a] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Security and Privacy '98*, pages 86–99. IEEE CS, 1998.

[ASW98b] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures (extended abstract). In *EUROCRYPT '98*, volume 1403 of *LNCS*, pages 591–606. Springer, 1998. Extended version in IEEE Journal on Selected Areas in Communications, 18(4): 593–610, 2000.

[Ate04] G. Ateniese. Verifiable encryption of digital signatures and applications. *ACM Trans. Inf. Syst. Secur.*, 7(1):1–20, 2004.

[AV04] G. Avoine and S. Vaudenay. Fair exchange with guardian angels. In *WISA '03*, volume 2908 of *LNCS*, pages 188–202. Springer, 2004.

[Bas99] D. Basin. Lazy infinite-state analysis of security protocols. In *CQRE '99*, volume 1740 of *LNCS*, pages 30–42. Springer, 1999.

[BCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *STOC '93*, pages 52–61. ACM Press, 1993.

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *CRYPTO '96*, volume 1109 of *LNCS*, pages 1–15. Springer, 1996.

[BCL+07] S. Blom, J. Calamé, B. Lisser, S. Orzan, J. Pang, J. van de Pol, M. Torabi Dashti, and A. Wijs. Distributed analysis with μCRL: A compendium of case studies. In *TACAS '07*, volume 4424 of *LNCS*, pages 683–689. Springer, 2007.

[BČMŠ05] L. Brim, I. Černá, P. Moravec, and J. Šimša. Distributed partial order reduction of state spaces. In *PDMC '04*, volume 128 of *ENTCS*, pages 63–74. Elsevier, 2005.

[BCT96] A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *WDAG '96*, volume 1151 of *LNCS*, pages 105–122. Springer, 1996.

[BDM98] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line TTP. In *IEEE Security and Privacy '98*, pages 77–85. IEEE CS, 1998.

[BDTW01] D. Boneh, X. Ding, G. Tsudik, and M. Wong. A method for fast revocation of public key certificates and security capabilities. In *10th Usenix Security Symposium*, pages 297–308. USENIX Association, 2001.

[BFG+01] S. Blom, W. Fokkink, J. Groote, I. van Langevelde, B. Lisser, and J. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *CAV '01*, volume 2102 of *LNCS*, pages 250–254, 2001.

[BFM03] M. Bugliesi, R. Focardi, and M. Maffei. Principles for entity authentication. In *PSI '03*, volume 2890 of *LNCS*, pages 294–307. Springer, 2003.

[BGG94] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *CRYPTO '94*, volume 839 of *LNCS*, pages 216–233. Springer, 1994.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88*, pages 1–10. ACM Press, 1988.

[BH99] L. Buttyán and J. Hubaux. Toward a formal model of fair exchange – a game theoretic approach. Technical Report SSC/1999/39, EPFL, Lausanne, Switzerland, Dec. 1999.

[BHC04] L. Buttyán, J. Hubaux, and S. Capkun. A formal model of rational exchange and its application to the analysis of Syverson's protocol. *J. Comput. Secur.*, 12(3-4):551–587, 2004.

[BK85] J. Bergstra and J. Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37:77–121, 1985.

[BK00] C. Boyd and P. Kearney. Exploring fair exchange protocols using specification animation. In *ISW '00*, volume 1975 of *LNCS*, pages 209–223. Springer, 2000.

[BLP03] G. Behrmann, K. Larsen, and R. Pelánek. To store or not to store. In *CAV '03*, volume 2725 of *LNCS*, pages 433–445. Springer, 2003.

[BLPW07] S. Blom, B. Lisser, J. van de Pol, and M. Weber. A database approach to distributed state space generation. In *PDMC '07*, 2007. to appear in ENTCS.

[Blu81] M. Blum. Three applications of the oblivious transfer: Part I: Coin flipping by the telephone; part II: How to exchange secrets; part III: How to send certified electronic mail. Technical report, Dept. EECS, University of California, Berkeley, 1981.

[BMV03] D. Basin, S Mödersheim, and L Viganò. CDiff: A new reduction technique for constraint-based analysis of security protocols. In *CCS '03*, pages 335–344. ACM Press, 2003.

[BOGMR90] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. *IEEE Trans. on Information Theory*, 36(1):40–46, 1990.

[Bol96] D. Bolignano. An approach to the formal verification of cryptographic protocols. In *CCS '96*, pages 106–118. ACM Press, 1996.

[Bor01] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *ICALP '01*, volume 2076 of *LNCS*, pages 667–681. Springer, 2001.

[BP01] G. Bella and L. Paulson. Mechanical proofs about a non-repudiation protocol. In *TPHOL '01*, volume 2152 of *LNCS*, pages 91–104. Springer, 2001.

[BP05] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Trans. Dependable Secur. Comput.*, 2(2):109–123, 2005.

[BP06] G. Bella and L. Paulson. Accountability protocols: Formalized and verified. *ACM Trans. Inf. Syst. Secur.*, 9(2):138–161, 2006.

[BPS01] J. Bergstra, A. Ponse, and S. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[BPSW02] M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial fairness and liveness. In *CSFW '02*, pages 160–174. IEEE CS, 2002.

[BS01] J. Bradfield and C. Stirling. Modal logics and mu-calculi: An introduction. In Bergstra et al. [BPS01], pages 293–330.

[BT94] A. Bahreman and D. Tygar. Certified electronic mail. In *NDSS '94*, pages 3–19. Internet Society, 1994.

[BVV84] M. Blum, U. Vazirani, and V. Vazirani. Reducibility among protocols. In *CRYPTO '83*, pages 137–146. Plenum pub., 1984.

[CA-01] CERT® Advisory CA-2001-09. Statistical weakness in TCP/IP initial sequence numbers, 2001. `http://www.cert.org/advisories/CA-2001-09.html`.

[Can02] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01*, pages 136–145. IEEE CS, 2002.

[Car94] U. Carlsen. Cryptographic protocols flaws. In *CSFW '94*, pages 192–200. IEEE CS, 1994.

[CCD88] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *STOC '88*, pages 11–19. ACM Press, 1988.

[CCT05] J. Cederquist, R. Corin, and M. Torabi Dashti. On the quest for impartiality: Design and analysis of a fair non-repudiation protocol. In *ICICS '05*, volume 3783 of *LNCS*, pages 27–39. Springer, 2005. Extended version as technical report 05-32, CTIT, University of Twente, The Netherland, Dec. 2005, `http://www.ub.utwente.nl/webdocs/ctit/1/00000138.pdf`.

[CDL06] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *J. Comput. Secur.*, 14(1):1–43, 2006.

[CE02] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *SAS '02*, volume 2477 of *LNCS*, pages 326–341. Springer, 2002.

[CEF03] B. Crispo, S. Etalle, and W. Fokkink. Accountability in electronic commerce protocols (ACCOUNT), 2003. Research proposal computer science open competition.

[CGP00] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[Cha03] R. Chadha. *A formal analysis of exchange of digital signatures*. PhD thesis, University of Pennsylvania, 2003.

[Che98] L. Chen. Efficient fair exchange with verifiable confirmation of signatures. In *ASIACRYPT '98*, volume 1514 of *LNCS*, pages 286–299. Springer, 1998.

[CIK+06] C. Chong, S. Iacob, P. Koster, J. Montaner, and R. van Buuren. License transfer in OMA-DRM. In *ESORICS '06*, volume 4189 of *LNCS*, pages 81–96. Springer, 2006.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature (version 1.0), 1997. `citeseer.ist.psu.edu/clark97survey.html`.

[CJM98] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *PROCOMET '98*, volume 125 of *IFIP*, pages 87–106. Chapman & Hall, 1998.

[CJM00a] E. Clarke, S. Jha, and W. Marrero. Partial order reductions for security protocol verification. In *TACAS '00*, volume 1785 of *LNCS*, pages 503–518. Springer, 2000.

[CJM00b] E. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Trans. Softw. Eng. Methodol.*, 9(4):443–487, 2000.

[CJM03] E. Clarke, S. Jha, and W. Marrero. Efficient verification of security protocols using partial-order reductions. *STTT*, 4(2):173–188, 2003.

[CKP04] L. Chen, C. Kudla, and K. Paterson. Concurrent signatures. In *EUROCRYPT '04*, volume 3027 of *LNCS*, pages 287–305. Springer, 2004.

[CKR+03] Y. Chevalier, R. Küsters, M Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao intruder for analyzing an unbounded number of sessions. Technical Report 4869, INRIA, France, Jul. 2003.

[CKRT03] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with xor. In *LICS '03*, pages 261–270. IEEE CS, 2003.

[CKS04] R. Chadha, S. Kremer, and A. Scedrov. Formal analysis of multi-party contract signing. In *CSFW '04*, pages 266–265. IEEE CS, 2004.

[CKW07] V. Cortier, R. Küsters, and B. Warinschi. A cryptographic model for branching time security properties – the case of contract signing protocols. In *ESORICS '07*, volume 4734 of *LNCS*, pages 422–437. Springer, 2007.

[Cle90] R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *CRYPTO '89*, volume 435 of *LNCS*, pages 573–588. Springer, 1990.

[CLS02] H. Comon-Lundh and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *J. of Telecomm. and Information Tech.*, 4:3–13, 2002.

[CLS03] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *LICS '03*, pages 271–280. IEEE CS, 2003.

[CM05] C. Cremers and S. Mauw. Checking secrecy by means of partial order reduction. In *SAM '04*, volume 3319 of *LNCS*, pages 177–194. Springer, 2005.

[CMSS03] R. Chadha, J. Mitchell, A. Scedrov, and V. Shmatikov. Contract signing, optimism, and advantage. In *CONCUR '03*, volume 2761 of *LNCS*, pages 366–382. Springer, 2003.

[COPT07] T. Chothia, S. Orzan, J. Pang, and M. Torabi Dashti. A framework for automatically checking anonymity with μCRL. In *TGC '06*, volume 4661 of *LNCS*, pages 301–318. Springer, 2007.

[CPT07] T. Chothia, J. Pang, and M. Torabi Dashti. Keeping secrets in resource aware components. In *QAPL '07*, volume 190 of *ENTCS*, pages 79–94. Elsevier, 2007.

[Cro94] L. Crowl. How to measure, present, and compare parallel performance. *IEEE Parallel and Distributed Technology*, 2(1):9–25, 1994.

[CT04] J. Cederquist and M. Torabi Dashti. Formal analysis of a fair payment protocol. In *Formal Aspect of Security and Trust*, volume 173 of *IFIP*, pages 41–54. Springer, 2004. Extended version as technical report SEN-R0410, CWI, Amsterdam, The Netherlands, Jul. 2004, `http://www.cwi.nl/ftp/CWIreports/SEN/SEN-R0410.pdf`.

[CT05] J. Cederquist and M. Torabi Dashti. An intruder model for verifying termination in security protocols. Technical Report 05-29, CTIT, University of Twente, Enschede, The Netherlands, Dec. 2005. `http://eprints.eemcs.utwente.nl/723/`.

[CT06] J. Cederquist and M. Torabi Dashti. An intruder model for verifying liveness in security protocols. In *FMSE '06*, pages 23–32. ACM Press, 2006.

[CTM07] J. Cederquist, M. Torabi Dashti, and S. Mauw. A certified email protocol using key chains. In *SSNDS '07*, pages 525–530. IEEE CS, 2007.

[CTS95] B. Cox, J. Tygar, and M. Sirbu. NetBill security and transaction protocol. In *1st Usenix Workshop in Electronic Commerce*, pages 77–88. USENIX Association, 1995.

[CV02] Y. Chevalier and L. Vigneron. Automated unbounded verification of security protocols. In *CAV '02*, volume 2404 of *LNCS*, pages 324–337. Springer, 2002.

[Dae98] J. Daemen. Management of secret keys: Dynamic key handling. In *State of the Art in Applied Cryptography*, volume 1528 of *LNCS*, pages 264–276. Springer, 1998.

[Den99] D. Denning. The limits of formal security models, Oct. 1999. National Computer systems security award acceptance speech, `http://www.cs.georgetown.edu/~denning/infosec/award.html`.

[DGLW96] R. Deng, L. Gong, A. A. Lazar, and W. Wang. Practical protocols for certified electronic mail. *J. Network Syst. Manage.*, 4(3):279–297, 1996.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, 1976.

[DH95] Y. Ding and P. Horster. Undetectable on-line password guessing attacks. *SIGOPS Oper. Syst. Rev.*, 29(4):77–86, 1995.

[DJH07] Y. Dodis, P. Joong Lee, and D. Hyun Yum. Optimistic fair exchange in a multi-user setting. In *PKC '07*, volume 4450 of *LNCS*, pages 118–133. Springer, 2007.

[DLM82] R. DeMillo, N. Lynch, and M. Merritt. Cryptographic protocols. In *STOC '82*, pages 383–400. ACM Press, 1982.

[DM05] P. Drielsma and S. Mödersheim. The ASW protocol revisited: A unified view. *ENTCS*, 125(1):145–161, 2005.

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[DR03] Y. Dodis and L. Reyzin. Breaking and repairing optimistic fair exchange from PODC 2003. In *DRM '03*, pages 47–54. ACM Press, 2003.

[DS97] B. Dutertre and S. Schneider. Using a PVS embedding of CSP to verify authentication protocols. In *TPHOL '97*, volume 1275 of *LNCS*, pages 121–136. Springer, 1997.

[DY81] D. Dolev and A. Yao. On the security of public key protocols (extended abstract). In *FOCS '81*, pages 350–357. IEEE CS, 1981.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, IT-29(2):198–208, 1983.

[EG83] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. Technical Report 285, Computer Science Dept., Technion, Haifa, Isreal, Jun. 1983.

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[EJ01] D. Eastlake and P. Jones. US secure hash algorithm 1 (SHA1), 2001. RFC 3174.

[EL86] E. Emerson and C. Lei. Temporal reasoning under generalized fairness constraints. In *STACS '86*, volume 210 of *LNCS*, pages 21–36. Springer, 1986.

[Eme90] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of theoretical computer science (vol. B): Formal models and semantics*, pages 995–1072. MIT Press, 1990.

[Eng97] R. Engelschall. URL rewriting engine, 1997. Apache HTTP server version 1.3.

[ES05a] N. Evans and S. Schneider. Verifying security protocols with PVS: Widening the rank function approach. *J. Logic and Algebraic Programming*, 64(2):253–284, 2005.

[ES05b] P. Ezhilchelvan and S. Shrivastava. A family of trusted third party based fair-exchange protocols. *IEEE Trans. Dependable Secur. Comput.*, 2(4):273–286, 2005.

[Eve83] S. Even. A protocol for signing contracts. *SIGACT News*, 15(1):34–39, 1983.

[EY80] S. Even and Y. Yacobi. Relations among public key signature systems. Technical Report 175, Computer Science Dept., Technion, Haifa, Isreal, Mar. 1980.

[FA05] K. Frikken and M. Atallah. Achieving fairness in private contract negotiation. In *FC '05*, volume 3570 of *LNCS*, pages 270–284. Springer, 2005.

[FE03] K. Fujimura and D. Eastlake. Requirements and design for voucher trading system (VTS). RFC 3506 (Informational), 2003.

[FFD+06] M. Fort, F. Freiling, L. Draque Penso, Z. Benenson, and D. Kesdogan. TrustedPals: Secure multiparty computation implemented with smart cards. In *ESORICS '06*, volume 4189 of *LNCS*, pages 34–48. Springer, 2006.

[FG01] R. Focardi and R. Gorrieri. Classification of security properties (part I: Information flow). In *FOSAD '00*, volume 2171 of *LNCS*, pages 331–396. Springer, 2001.

[FGK+96] J. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP - a protocol validation and verification toolbox. In *CAV '96*, volume 1102 of *LNCS*, pages 437 – 440. Springer, 1996.

[FGY92] M. Franklin, Z. Galil, and M. Yung. An overview of secure distributed computing. Technical Report TR CUCS-008-92, Dep. of Computer Science, Columbia University, Mar. 1992.

[FKT+99] K. Fujimura, H. Kuno, M. Terada, K. Matsuyama, Y. Mizuno, and J. Sekine. Digital-ticket-controlled digital ticket circulation. In *Proc. the 8th USENIX Security Symposium*, pages 229 – 240. USENIX Association, 1999.

[FLM86] M. Fischer, N. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distrib. Comput.*, 1(1):26–39, 1986.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[Fok07] W. Fokkink. *Modelling Distributed Systems*. Texts in Theoretical Computer Science. Springer, 2007.

[FPH00] J. Ferrer-Gomila, M. Payeras-Capellà, and L. Huguet-i-Rotger. An efficient protocol for certified electronic mail. In *ISW '00*, volume 1975 of *LNCS*, pages 237–248. Springer, 2000.

[FPH02] J. Ferrer-Gomila, M. Payeras-Capellà, and L. Huguet-i-Rotger. A realistic protocol for multi-party certified electronic mail. In *ISC '02*, volume 2433 of *LNCS*, pages 210–219. Springer, 2002.

[FPH04] J. Ferrer-Gomila, M. Payeras-Capellà, and L. Huguet-i-Rotger. Optimality in asynchronous contract signing protocols. In *TrustBus '04*, volume 3184 of *LNCS*, pages 200–208. Springer, 2004.

[FR97] M. Franklin and M. Reiter. Fair exchange with a semi-trusted third party (extended abstract). In *CCS '97*, pages 1–5. ACM Press, 1997.

[Fra86] N. Francez. *Fairness*. Springer, 1986.

[FTW07] W. Fokkink, M. Torabi Dashti, and A. Wijs. Partial order reduction for branching security protocols. In *WITS '07*, pages 178–193, 2007.

[GHK+07] R. Guerraoui, M. Herlihy, P. Kouznetsov, N. Lynch, and C. Newport. On the weakest failure detector ever. In *PODC '07*, pages 235–243. ACM Press, 2007.

[Gia99] D. Giannakopoulou. *Model Checking for Concurrent Software Architectures*. PhD thesis, Imperial College of Science Technology and Medicine, University of London, 1999.

[GJM99] J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO '99*, volume 1666 of *LNCS*, pages 449–466. Springer, 1999.

[GL02a] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *DISC '02*, volume 2508 of *LNCS*, pages 17–32. Springer, 2002.

[GL02b] J. Groote and B. Lisser. Computer assisted manipulation of algebraic process specifications. *SIGPLAN Not.*, 37(12):98–107, 2002. Extended version as technical report SEN-R0117, CWI, Amsterdam, The Netherlands, May 2001, `ftp://ftp.cwi.nl/pub/CWIreports/SEN/SEN-R0117.pdf`.

[Gla93] R. van Glabbeek. The linear time - branching time spectrum II. In *CONCUR '93*, volume 715 of *LNCS*, pages 66–81. Springer, 1993.

[Gli07] V. Gligor. On the evolution of adversary models in security protocols: From the beginning to sensor networks. In *ASIACCS '07*, page 3. ACM Press, 2007.

[Gol05] O. Goldreich. Foundations of cryptography: A primer. *Found. Trends Theor. Comput. Sci.*, 1(1):1–116, 2005.

[Gol06a] O. Goldreich. On post-modern cryptography, 2006. `http://eprint.iacr.org/2006/461`.

[Gol06b] D. Gollmann. Why trust is bad for security. *ENTCS*, 157(3):3–9, 2006.

[Gon05] N. González-Deleito. *Trust relationships in exchange protocols*. PhD thesis, Université Libre de Bruxelles, 2005.

[GP95] J. Groote and A. Ponse. The syntax and semantics of μCRL. In *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62. Springer, 1995. Also as technical report CS-R9076, CWI, Amsterdam, The Netherlands, Dec. 1990.

[GR01] J. Groote and M. Reniers. Algebraic process verification. In Bergstra et al. [BPS01], pages 1151–1208.

[GR03] S. Gürgens and C. Rudolph. Security analysis of (un-) fair non-repudiation protocols. In *FASec '02*, volume 2629 of *LNCS*, pages 97–114. Springer, 2003.

[GR06] B. Garbinato and I. Rickebusch. A topological condition for solving fair exchange in Byzantine environments. In *ICICS '06*, volume 4307 of *LNCS*, pages 30–49. Springer, 2006.

[Gra78] J. Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, volume 60 of *LNCS*, pages 393–481. Springer, 1978.

[GRV05] S. Gürgens, C. Rudolph, and H. Vogt. On the security of fair non-repudiation protocols. *Int. J. Inf. Sec.*, 4(4):253–262, 2005.

[HJP05] Y. Hu, M. Jakobsson, and A. Perrig. Efficient constructions for one-way hash chains. In *ACNS '05*, volume 3531 of *LNCS*, pages 423–441, 2005.

[HLS03] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *J. Comput. Secur.*, 11(2):217–244, 2003.

[HM84] J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In *PODC '84*, pages 50–61. ACM Press, 1984.

[HS04] H. Hüttel and J. Srba. Recursion vs. replication in simple cryptographic protocols. Technical Report RS-04-23, BRICS, University of Aarhus, Denmark, Oct. 2004.

[HS06] J. Heather and S. Schneider. To infinity and beyond or, avoiding the infinite in security protocol analysis. In *SAC '06*, pages 346–353. ACM Press, 2006.

[HT96] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Trans. Softw. Eng.*, 22(1):16–30, 1996.

[HTWC96] N. Heintze, J. Tygar, J. Wing, and H. Chi Wong. Model checking electronic commerce protocols. In *2nd Usenix Workshop in Electronic Commerce*, pages 10–10. USENIX Association, 1996.

[Hüt02] H. Hüttel. Deciding framed bisimulation. Technical Report RS-02-25, BRICS, University of Aarhus, Denmark, May 2002.

[HW06] M. Hammer and M. Weber. "To Store or Not To Store" reloaded: Reclaiming memory on demand. In *FMICS '06*, volume 4346 of *LNCS*, pages 51–66. Springer, 2006.

[IIK02] C. Ito, M. Iwaihara, and Y. Kambayashi. Fair exchange under limited trust. In *TES '02*, volume 2444 of *LNCS*, pages 161–170. Springer, 2002.

[IZS05] K. Imamoto, J. Zhou, and K. Sakurai. An evenhanded certified email system for contract signing. In *ICICS '05*, volume 3783 of *LNCS*, pages 1–13. Springer, 2005.

[Jak95] M. Jakobsson. Ripping coins for fair exchange. In *EUROCRYPT '95*, volume 921 of *LNCS*, pages 220–230. Springer, 1995.

[JHG99] F. Javier Thayer Fabrega, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *J. Comput. Secur.*, 7(2-3):191–230, 1999.

[JY96] M. Jakobsson and M. Yung. Revokable and versatile electronic money. In *CCS '96*, pages 76–87. ACM Press, 1996.

[Ker83] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, 161–191, 1883.

[KG91] R. Kailar and V. Gligor. On belief evolution in authentication protocols. In *CSFW '91*, pages 103–116. IEEE CS, 1991.

[KH06] H. Khurana and H. Hahm. Certified mailing lists. In *ASIACCS '06*, pages 46–58. ACM Press, 2006.

[KK05] D. Kähler and R. Küsters. Constraint solving for contract-signing protocols. In *CONCUR '05*, volume 3653 of *LNCS*, pages 233–247. Springer, 2005.

[KKT07] D. Kähler, R. Küsters, and T. Truderung. Infinite state AMC-model checking for cryptographic protocols. In *LICS '07*, pages 181–192. IEEE CS, 2007. Extended version as technical report 0702, Christian-Albrechts-Universität Kiel, Germany, Feb. 2007.

[KKW05] D. Kähler, R. Küsters, and T. Wilke. Deciding properties of contract-signing protocols. In *STACS '05*, volume 3404 of *LNCS*, pages 158–169. Springer, 2005. Extended version as technical report 0409, Christian-Albrechts-Universität Kiel, Germany, Sept. 2004.

[KM00] S. Kremer and O. Markowitch. A multi-party non-repudiation protocol. In *IFIP Working Conf. on Information Security for Global Information Infrastructures*, pages 271–280. Kluwer, 2000.

[KM01] S. Kremer and O. Markowitch. Selective receipt in certified email. In *INDOCRYPT '01*, volume 2247 of *LNCS*, pages 136–148. Springer, 2001.

[KMZ02] S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, 2002.

[Kob07] N. Koblitz. The uneasy relationship between mathematics and cryptography. *Notices of the AMS*, 54(8):972–979, 2007.

[KR01] S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *CONCUR '01*, volume 2154 of *LNCS*, pages 551–565. Springer, 2001.

[Kre03] S. Kremer. *Formal Analysis of Optimistic Fair Exchange Protocols*. PhD thesis, Université Libre de Bruxelles, 2003.

[KS03] T. Kempster and C. Stirling. Modeling and model checking mobile phone payment systems. In *FORTE '03*, volume 2767 of *LNCS*, pages 95–110. Springer, 2003.

[KSW98] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In *Security Protocols Workshop '97*, volume 1361 of *LNCS*, pages 91–104. Springer, 1998.

[Lam81] L. Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.

[Laz99] R. Lazic. *A semantic study of data independence with applications to model checking*. PhD thesis, Oxford University, 1999.

[LBL+99] G. Leduc, O. Bonaventure, L. Léonard, E. Koerner, and C. Pecheur. Model-based verification of a security protocol for conditional access to services. *Form. Methods Syst. Des.*, 14(2):171–191, 1999.

[LM05] C. Lynch and C. Meadows. On the relative soundness of the free algebra model for public key encryption. *ENTCS*, 125(1):43–54, 2005.

[LMS02] F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *LICS '02*, pages 383–392. IEEE CS, 2002.

[LNJ01] P. Liu, P. Ning, and S. Jajodia. Avoiding loss of fairness owing to failures in fair data exchange systems. *Decision Support Systems*, 31(3):337–350, 2001.

[Lou00] P. Louridas. Some guidelines for non-repudiation protocols. *SIGCOMM Comput. Commun. Rev.*, 30(5):29–38, 2000.

[Low97] G. Lowe. A hierarchy of authentication specifications. In *CSFW '97*, page 31. IEEE CS, 1997.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[LYH04] Y. Li, W. Yang, and C. Huang. Preventing type flaw attacks on security protocols with a simplified tagging scheme. In *ISICT '04*, pages 244–249. Trinity College Dublin, 2004.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[McM00] K. McMillan. The SMV system for SMV version 2.5.4, Nov. 2000. `http://www.cs.cmu.edu/~modelcheck/smv/smvmanual.ps`.

[MD02] J. Monteiro and R. Dahab. An attack on a protocol for certified delivery. In *ISC '02*, volume 2433 of *LNCS*, pages 428–436. Springer, 2002.

[Mea03] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communications*, 21(1):44–54, 2003.

[Mea04] C. Meadows. Ordering from satan's menu: A survey of requirements specification for formal analysis of cryptographic protocols. *Sci. Comput. Program.*, 50(1-3):3–22, 2004.

[MHL+92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Trans. Database Syst.*, 17(1):94–162, 1992.

[Mic97] S. Micali. Certified email with invisible post offices, 1997. Presented at RSA Security Conference.

[Mic03] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC '03*, pages 12–19. ACM Press, 2003.

[Mil03] J. Millen. On the freedom of decryption. *Inf. Process. Lett.*, 86(6):329–333, 2003.

[MK01] O. Markowitch and S. Kremer. An optimistic non-repudiation protocol with transparent trusted third party. In *ISC '01*, volume 2200 of *LNCS*, pages 363–378. Springer, 2001.

[MKR05] A. Mukhamedov, S. Kremer, and E. Ritter. Analysis of a multi-party fair exchange protocol and formal proof of correctness in the strand space model. In *FC '05*, volume 3570 of *LNCS*, pages 255–269. Springer, 2005.

[MMS97] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Security and Privacy '97*, pages 141–153. IEEE CS, 1997.

[MP71] R. McNaughton and S. Papert. *Counter-Free Automata (M.I.T. research monograph no. 65)*. MIT Press, 1971.

[MR99] O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In *Proc. 2nd Workshop on Security in Communication Network*, 1999.

[MR06] A. Mukhamedov and M. Ryan. Resolve-impossibility for a contract-signing protocol. In *CSFW '06*, pages 167–176. IEEE CS, 2006.

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS '01*, pages 166–175. ACM Press, 2001.

[MS02] O. Markowitch and S. Saeednia. Optimistic fair-exchange with transparent signature recovery. In *FC '01*, volume 2339 of *LNCS*, pages 339 – 350. Springer, 2002.

[MS03] R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Sci. Comput. Program.*, 46(3):255–281, 2003.

[MVO96] A. Menezes, S. Vanstone, and P. van Oorschot. *Handbook of Applied Cryptography*. CRC Press, 1996.

[MW87] S. Moran and Y. Wolfstahl. Extended impossibility results for asynchronous complete networks. *Inf. Process. Lett.*, 26(3):145–151, 1987.

[Nen05] A. Nenadić. *A security solution for fair exchange and non-repudiation in e-commerce*. PhD thesis, University of Manchester, 2005.

[NIS01] NIST. Advanced encryption standard (AES), 2001. FIPS PUB 197.

[NPG+05] S. Nair, B. Popescu, C. Gamage, B. Crispo, and A. Tanenbaum. Enabling DRM-preserving digital content redistribution. In *7th IEEE Conf. E-Commerce Technology*, pages 151–158. IEEE CS, 2005.

[NTCT07] S. Nair, M. Torabi Dashti, B. Crispo, and A. Tanenbaum. A hybrid PKI-IBC based ephemerizer system. In *IFIP SEC '07*, volume 232 of *IFIP*, pages 241–252. Springer, 2007.

[NV95] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *J. ACM*, 42(2):458–487, 1995.

[NZB04] A. Nenadić, N. Zhang, and S. Barton. Fair certified email delivery. In *SAC '04*, pages 391–396. ACM Press, 2004.

[Pan00] S. Pancho. Paradigm shifts in protocol analysis. In *New Security Paradigms Workshop '99*, pages 70–79. ACM Press, 2000.

[Pan02] J. Pang. Analysis of a security protocol in μCRL. In *ICFEM '02*, volume 2495 of *LNCS*, pages 396–400. Springer, 2002.

[Pau97] L. Paulson. Mechanized proofs for a recursive authentication protocol. In *CSFW '97*, pages 84–94. IEEE CS, 1997.

[Pau98] L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.*, 6(1-2):85–128, 1998.

[PCS03] J. Park, E. Chong, and H. Siegel. Constructing fair-exchange protocols for e-commerce via distributed computation of RSA signatures. In *PODC '03*, pages 172–181. ACM Press, 2003.

[Pel93] D. Peled. All from one, one for all: On model checking using representatives. In *CAV '93*, volume 697 of *LNCS*, pages 409–423. Springer, 1993.

[Pel97] D. Peled. Partial order reduction: Linear and branching temporal logics and process algebras. In Peled et al. [PPH97], pages 233–257.

[Pel98] D. Peled. Ten years of partial order reduction. In *CAV '98*, volume 1427 of *LNCS*, pages 17–28. Springer, 1998.

[Pel04] R. Pelánek. Typical structural properties of state spaces. In *SPIN '04*, volume 2989 of *LNCS*, pages 5–22. Springer, 2004.

[PG99] H. Pagnia and F. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Department of Computer Science, Darmstadt University of Technology, Darmstadt, Germany, Mar. 1999.

[PG02] R. Palmer and G. Gopalakrishnan. A distributed partial order reduction algorithm. In *FORTE '02*, volume 2529 of *LNCS*, page 370. Springer, 2002.

[PG06] S. Pancho-Festin and D. Gollmann. On the formal analyses of the Zhou-Gollmann non-repudiation protocol. In *FAST '05*, volume 3866 of *LNCS*, pages 5–15. Springer, 2006.

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS '77*, pages 46–57. IEEE, 1977.

[Pol01] J. van de Pol. Just-in-time: On strategy annotations. Technical Report SEN-R0105, CWI, Amsterdam, The Netherlands, Mar. 2001. `http://www.cwi.nl/ftp/CWIreports/SEN/SEN-R0105.pdf`.

[PPH97] D. Peled, V. Pratt, and G. Holzmann, editors. *Partial order methods in verification*, volume 29 of *DIMACS series in dicrete mathematics and theoretical computer science*. AMS Press, 1997.

[Pra94] A. Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Asp. Comput.*, 6(5):495–512, 1994.

[PSW98] B. Pfitzmann, M. Schunter, and M. Waidner. Optimal efficiency of optimistic contract signing. In *PODC '98*, pages 113–122. ACM Press, 1998. Extended version as technical report RZ 2994 (#93040), IBM Zürich Research Lab, Feb. 1998.

[PTSC00] A. Perrig, J. Tygar, D. Song, and R. Canetti. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Security and Privacy '00*, page 56. IEEE CS, 2000.

[PV04] J. van de Pol and M. Valero Espada. Modal abstractions in μCRL. In *AMAST '04*, volume 3116 of *LNCS*, pages 409–425. Springer, 2004.

[PVG03] H. Pagnia, H. Vogt, and F. Gärtner. Fair exchange. *The Computer Journal*, 46(1):55–7, 2003.

[PW02] R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *CSFW '02*, pages 282–294. IEEE CS, 2002.

[QS83] J. Queille and J. Sifakis. Fairness and related properties in transition systems - a temporal logic to deal with fairness. *Acta Inf.*, 19:195–220, 1983.

[Rab81] M. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken computation Lab, Harvard University, May 1981.

[Rab83] M. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983.

[RG97] A. Roscoe and M. Goldsmith. The perfect "spy" for model-checking cryptoprotocols. In *Proc. of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. `http://dimacs.rutgers.edu/Workshops/Security/program.html`.

[Ric53] H. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74:358–366, 1953.

[Riv92] R. Rivest. The MD5 message-digest algorithm, 1992. RFC 1321.

[RR00] I. Ray and I. Ray. An optimistic fair exchange e-commerce protocol with automated dispute resolution. In *EC-WEB '00*, volume 1875 of *LNCS*, pages 84–93. Springer, 2000.

[RRN05] I. Ray, I. Ray, and N. Natarajan. An anonymous and failure resilient fair-exchange e-commerce protocol. *Decision Support Systems*, 39(3):267–292, 2005.

[RS98a] J. Riordan and B. Schneier. A certified email protocol. In *ACSAC '98*, pages 347–352. IEEE CS, 1998.

[RS98b] P. Ryan and S. Schneider. An attack on a recursive authentication protocol. A cautionary tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[RSG+00] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: The CSP Approach*. Addison-Wesley, 2000.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *CSFW '01*, page 174. IEEE CS, 2001.

[San97] T. Sandholm. Unenforced e-commerce transactions. *IEEE Internet Computing*, 1(6):47–54, 1997.

[Sch98] S. Schneider. Formal analysis of a non-repudiation protocol. In *CSFW '98*, pages 54–65. IEEE CS, 1998.

[Sch00] M. Schunter. *Optimistic fair exchange*. PhD thesis, Universität des Saarlandese, 2000.

[SM00] P. Syverson and C. Meadows. Dolev-Yao is no better than Machiavelli. In *WITS '00*, pages 87–92, 2000.

[SM02] V. Shmatikov and J. Mitchell. Finite-state analysis of two contract signing protocols. *Theor. Comput. Sci.*, 283(2):419–450, 2002.

[SMZ04] W. Susilo, Y. Mu, and F. Zhang. Perfect concurrent signature schemes. In *ICICS '04*, volume 3269 of *LNCS*, pages 14–26. Springer, 2004.

[SS98] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *CSFW '98*, pages 106–115. IEEE CS, 1998.

[Ste76] N. Stenning. A data transfer protocol. *Computer Networks*, 1(2):99–110, 1976.

[SW02] T. Sandholm and X. Wang. (Im)possibility of safe exchange mechanism design. In *8th national conference on artificial intelligence*, pages 338–344. AAAI, 2002.

[SWZ06] M. Shao, G. Wang, and J. Zhou. Some common attacks against certified email protocols and the countermeasures. *Computer Communications*, 29(15):2759–2769, 2006.

[SXL05] M. Srivatsa, L. Xiong, and L. Liu. ExchangeGuard: A distributed protocol for electronic fair-exchange. In *IPDPS '05*, page 105b. IEEE CS, 2005.

[Syv94] P. Syverson. A taxonomy of replay attacks. In *CSFW '94*, pages 187–191. IEEE CS, 1994.

[Syv97] P. Syverson. A different look at secure distributed computation. In *CSFW '97*, pages 109–115. IEEE CS, 1997.

[Syv98] P. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *CSFW '98*, pages 2–13. IEEE CS, 1998.

[SZW05] M. Shao, J. Zhou, and G. Wang. On the security of a certified email scheme with temporal authentication. In *ICCSA '05*, volume 3482 of *LNCS*, pages 701–710. Springer, 2005.

[Tan96a] A. Tanenbaum. *Computer Networks*. Prentice Hall, 3rd edition, 1996.

[Tan96b] L. Tang. Verifiable transaction atomicity for electronic payment protocols. In *ICDCS '96*, pages 261–269. IEEE CS, 1996.

[TEB06] F. Tiplea, C. Enea, and C. Birjoveanu. Decidability and complexity results for security protocols. In *VISSAS '05*, volume 1 of *NATO Security through Science Series D: Information and Communication Security*, pages 185–211. IOS Press, 2006.

[TIHF04] M. Terada, M. Iguchi, M. Hanadate, and K. Fujimura. An optimistic fair exchange protocol for trading electronic rights. In *CARDIS '04*, pages 255–270. Kluwer, 2004.

[TKJ07] M. Torabi Dashti, S. Krishnan Nair, and H. Jonker. Nuovo DRM paradiso: Towards a verified fair DRM scheme. In *FSEN '07*, volume 4767 of *LNCS*, pages 33–48. Springer, 2007. Extended version as technical report SEN-R0602, CWI, Amsterdam, The Netherlands, Mar. 2006, `ftp.cwi.nl/CWIreports/SEN/SEN-R0602.pdf`.

[TMH06] M. Terada, K. Mori, and S. Hongo. An optimistic NBAC-based fair exchange method for arbitrary items. In *CARDIS '06*, volume 3928 of *LNCS*, pages 105–118. Springer, 2006.

[TMI+06] M. Terada, K. Mori, K. Ishii, S. Hongo, T. Usaka, N. Koshizuka, and K. Sakamura. A framework for distributed inter-smartcard communication. *IPSJ Digital Courier*, 2:120–132, 2006.

[Tor03] M. Torabi Dashti. *Formal verification of cryptographic protocols*. MSc thesis, Sharif University of Technology, 2003. *(in Persian)*.

[TSS06] D. Tonien, W. Susilo, and R. Safavi-Naini. Multi-party concurrent signatures. In *ISC '06*, volume 4176 of *LNCS*, pages 131–145. Springer, 2006.

[TW07a] M. Torabi Dashti and Y. Wang. Risk balance in exchange protocols. In *Asian '07*, volume 4846 of *LNCS*, pages 70–77. Springer, 2007.

[TW07b] M. Torabi Dashti and A. Wijs. Pruning state spaces with extended beam search. In *ATVA '07*, volume 4762 of *LNCS*, pages 543–552. Springer, 2007.

[TWL07] M. Torabi Dashti, A. Wijs, and B. Lisser. Distributed partial order reduction for security protocols. In *PDMC '07*, 2007. to appear in ENTCS.

[Tyg96] J. Tygar. Atomicity in electronic commerce. In *PODC '96*, pages 8–26. ACM Press, 1996.

[Vog03] H. Vogt. Asynchronous optimistic fair exchange based on revocable items. In *FC '03*, volume 2742 of *LNCS*, pages 208–222. Springer, 2003.

[VPG01] H. Vogt, H. Pagnia, and F. Gärtner. Using smart cards for fair exchange. In *WELCOM '01*, volume 2232 of *LNCS*, pages 101–113. Springer, 2001.

[VV06] D. Varacca and H. Völzer. New perspectives on fairness. *Bulletin of the EATCS*, 90:90–108, 2006.

[VW86] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS '86*, pages 332–344. IEEE CS, 1986.

[WBZ04] G. Wang, F. Bao, and J. Zhou. On the security of a certified email scheme. In *INDOCRYPT '04*, volume 3348 of *LNCS*, pages 48–60. Springer, 2004.

[WBZ06] G. Wang, F. Bao, and J. Zhou. The fairness of perfect concurrent signatures. In *ICICS '06*, volume 4307 of *LNCS*, pages 435–451. Springer, 2006.

[WH07] K. Wei and J. Heather. A theorem-proving approach to verification of fair non-repudiation protocols. In *FAST '06*, volume 4691 of *LNCS*, pages 202–219. Springer, 2007.

[Wij07] A. Wijs. *What to Do Next? Analysing and optimising system behaviour in time*. PhD thesis, Vrije Universiteit Amsterdam, 2007.

[WL93] T. Woo and S. Lam. A semantic model for authentication protocols. In *IEEE Security and Privacy '93*, pages 178–194. IEEE CS, 1993.

[Wou01] A. Wouters. Manual for the μCRL tool set (version 2.8.2). Technical Report SEN-R0130, CWI, Amsterdam, The Netherlands, Dec. 2001.

[WYY05] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *CRYPTO '05*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.

[YC79] Y. Yemini and D. Cohen. Some issues in distributed processes communication. In *Proc. of the 1st International Conf. on Distributed Computing Systems*, pages 199–203, 1979.

[ZDB99] J. Zhou, R. Deng, and F. Bao. Evolution of fair non-repudiation with TTP. In *ACISP '99*, volume 1587 of *LNCS*, pages 258–269. Springer, 1999.

[ZDB00] J. Zhou, R. Deng, and F. Bao. Some remarks on a fair exchange protocol. In *PKC '00*, volume 1751 of *LNCS*, pages 46–57. Springer, 2000.

[ZG96a] J. Zhou and D. Gollmann. Certified electronic mail. In *ESORICS '96*, volume 1146 of *LNCS*, pages 160–171. Springer, 1996.

[ZG96b] J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *IEEE Security and Privacy '96*, pages 55–61. IEEE CS, 1996.

[ZG96c] J. Zhou and D. Gollmann. Observations on non-repudiation. In *ASIACRYPT '96*, volume 1163 of *LNCS*, pages 133–144. Springer, 1996.

[ZG97a] J. Zhou and D. Gollmann. An efficient non-repudiation protocol. In *CSFW '97*, pages 126–132. IEEE CS, 1997.

[ZG97b] J. Zhou and D. Gollmann. Evidence and non-repudiation. *J. Netw. Comput. Appl.*, 20(3):267–281, 1997.

[Zho04] J. Zhou. On the security of a multi-party certified email protocol. In *ICICS '04*, volume 3269 of *LNCS*, pages 40–52. Springer, 2004.

Samenvatting

Eerlijk duurt het langst
Ontwerp en formele verificatie van optimistische fair exchange protocollen

Dit proefschrift bestaat uit twee gedeeltes, namelijk het *ontwerp* en de *verificatie* van optimistische "fair exchange" protocollen.

Het eerste gedeelte draait om een nieuw gecertificeerd email-protocol, waarmee Alice een email naar Bob kan sturen, in ruil voor een ontvangstbewijs. Dit is een fair exchange protocol: Bob ontvangt de email dan en slechts dan als Alice het ontvangstbewijs krijgt. Een dergelijke uitwisseling is alleen mogelijk indien een vertrouwde derde partij bij het protocol betrokken is. De kracht van het protocol ligt in het gebruik van sleutel-ketens, waardoor deze derde partij minder opslagruimte nodig heeft om fairness te kunnen garanderen.

In het tweede gedeelte wordt een modellering van indringers, met een zorgvuldig ontworpen fairness beperking, ontwikkeld die het mogelijk maakt om liveness aspecten van optimistische fair exchange protocollen te verifiëren. Ons indringer-model is equivalent met het standaard Dolev-Yao indringer-model, behalve dat ieder bericht dat door een communicatiekanaal wordt verstuurd uiteindelijk zijn bestemming dient te bereiken. Dergelijke betrouwbare communicatiekanalen zijn cruciaal in de meeste optimistische fair exchange protocollen. Om op empirische wijze de effectiviteit van ons indringer-model aan te tonen, worden twee protocollen voor electronische betaling en voor digitale rechten formeel geanalyseerd op basis van dit model.

Verder wordt een bestaande "partiële ordening reductie" techniek uitgebreid, zodat deze techniek toepasbaar wordt op optimistische fair exchange protocollen. In deze protocollen hebben de deelnemers gedurende de uitwisseling gewoonlijk zekere keuzemomenten, die een speciale behandeling vereisen in de partiële ordening reductie techniek. De schaalbaarheid en effectiviteit van de techniek worden door middel van enkele case studies aangetoond.

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty

of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J.Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W.Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07